НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерна інженерія»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Планування на основі штучного інтелекту»

Виконав:
студент IV курсу, групи ІО-64
Кобилюк Андрій Григорович _____

Керівник:
к.т.н, доцент,
Волокита Артем Миколайович _____

Консультант з нормоконтролю:
д.т.н, професор
Сімоненко Валерій Павлович _____

Рецензент:
ст. викладач кафедри АСОІУ
Олійник Юрій Олександрович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту
Кобилюка Андрія Григоровича

1. Тема роботи «Планування на основі штучного інтелекту», керівник роботи Волокита Артем Миколайович, к.т.н, доцент, затверджені наказом по університету від *«07» травня 2020 р. №1081-с*

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи: *технічна документація, теоретичні та статистичні дані;*

4. Зміст роботи: *аналіз предметної області, вибір й обгронтування засобів реалізації, розробка програмного продукту, результати моделювання.*

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): *блок-схема роботи програми, діаграма класі програми, схема роботи програми.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	*д.т.н, проф. Сімоненко В. П.*		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	*Затвердження теми роботи*	*01.09.2019*	
2	*Вивчення та аналіз завдання*	*01.10.2020-31.12.2020*	
3	*Написання вступної частини та огляду предметної області*	*01.01.2020-15.03.2020*	
4	*Розробка архітектури системи*	*16.03.2020-10.04.2020*	
5	*Програмна реалізація системи*	*11.04.2020-30.04.2020*	
6	*Виправлення помилок*	*01.05.2020-10.05.2020*	
7	*Оформлення документації дипломної роботи*	*11.05.2020-25.05.2020*	
8	*Передзахист*	*26.05.2020*	
9	*Захист*	*18.06.2020*	

Студент Андрій КОБИЛЮК

Керівник Артем ВОЛОКИТА

АНОТАЦІЯ

В бакалаврській роботі розглядається принцип роботи планування, процес роботи якого базується на алгоритмах машинного навчання. Як практична сторона реалізований програмний продукт, який моделює його роботу.

Програма дозволяє отримати й оцінити характеристики віртуальної комп’ютерної системи для порівняння ефективності роботи планувальника у відношенні із іншими аналогами. Програмний продукт був створений на мові Python у візуальному середовищі Pycharm IDE Community Edition 2020.1.

ANNOTATION

The bachelor's work deals the principle of planning, the process of which is based on machine learning algorithms. As a practical side, there is a software product that models its work.

The program allows you to obtain and evaluate the characteristics of a virtual computer system to compare the performance of the scheduler against other peers. The software was created in Python in the Pycharm IDE Community Edition 2020.1. visual environment.

Відомість
до дипломного проєкту
на тему: «Планування на основі штучного інтелекту»

Київ – 2020 року

	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ екзем.	Примітка
			Документація загальна			
			Розроблена заново			
1	А4	ІАЛЦ.467200.001 ВП	Відомість до проєкту	1		
2	А4	ІАЛЦ.467200.002 ТЗ	Технічне завдання	4		
3	А4	ІАЛЦ.467200.003 ПЗ	Пояснювальна записка	69		
4	А4	ІАЛЦ.467200.004 Д1	Структурна схема роботи	1		
5	А4	ІАЛЦ.467200.005 Д2	Функціональна схема класів	1		
6	А4	ІАЛЦ.467200.006 Д3	Принципова схема роботи	1		
7	А4	ІАЛЦ.467200.007 Д4	Лістинг програми	15		

Зм.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467200.001 ВП		
Розроб.		Кобилюк А.Г.			Планування на основі штучного інтелекту Відомість до проєкту	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Волокита А.М.				Т	
Н. контр.		Сімоненко В.П.				НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІО-64	
Затв.		Волокита А.М.					

Технічне завдання
до дипломного проєкту
на тему: «Планування на основі штучного інтелекту»

Київ – 2020 року

ЗМІСТ

1. НАЙМЕНУВАННЯ І ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування: планування на основі штучного інтелекту.

Область застосування: альтернатива існуючим алгоритмам й програмам планування.

2. ПІДСТАВИ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітньогорівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут».

3. МЕТА І ПРИЗНАЧАННЯ РОЗРОБКИ

Метою даного проекту є розгляд принципів роботи планування на основі штучного інтелекту та розробка програмного продукту, який реалізовуватиме дану модель.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії та практики основ операційних систем, технічна документація, публікації в періодичних виданнях, довідники, публікації в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

- Відносно простий і інтуїтивно-зрозумілий інтерфейс системи.
- Можливість моделювання різних топологій систем й задач.
- Незалежність - система повинна бути автономна і не залежати відвстановленого програмного забезпечення, виключаючи платформу (операційну систему) а також не мати потреб в підключенні до Інтернету.
- Технічна коректність та актуальність інформації, що становить наповнення системи.

5.2. Вимоги до програмного забезпечення

- MS Windows 7/8/10 або Linux OS.
- Python 3.7 версія і вище.
- Браузер Google Chrome.

5.3. Вимоги до апаратної частини

- 32-розрядний (x86) або 64-розрядний (x64) процесор із тактовою частотою 1 ГГц або швидший.
- Оперативної памяті не менше 2 Гб.
- Вільний простір жосткого диска не менше 2 Гб (для програми й програмного забезпечення).

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	15.12.2019
Скадання і узгодження технічного завдання	25.01.2020
Створення модулів розроблюваної системи	20.02.2020
Тестування окремих модулів системи	15.03.2020
Доопрацювання, відладка і виправлення помилок	15.04.2020
Оформлення документації дипломної работи	19.05.2020

Пояснювальна записка
до дипломного проєкту
на тему: «Планування на основі штучного інтелекту»

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI	Artificial Intelligence– популярне англомовне скорочення для позначення «Штучного інтелекту».
API	Application programming interface -
GA	Genetic Algorithm – це спеціалізований алгоритм сфери машинного навчання.
GP	Genetic Programing – евристична техніка пошуку у сфері штучного інтелекту.
IDE	Integrated Development Environment – комплексне програмне рішення для розробки програмного забезпечення.
HTTP	HyperText Transfer Protocol – протокол «клієнт-сервер» взаємодії.
NN	Neural Network – є мережею або ланцюгом нейронів , або в сучасному сенсі, штучна нейронна мережа , що складається з штучних нейронів або вузлів.
RAM	Random Access Memory – пам'ять з довільною вибіркою.
SQL	Structured query language – сцеціалізована мова для роботи із реляційними базами даних.
UML	Unified Modeling Language – узагальнена мова моделювання, використовується у об'єктно-орієнтованому програмуванні.
ПЗ	Програмне забезпечення.

ВСТУП

У теперішні роки штучний інтелект ефективно застосовується у різноманітних сферах людського життя [8]: у ролі цифрового асистента, для перекладу іноземного тексту, для автоматичного керування транспортними засобами тощо. Із приходом ери масової «діджиталізації» викикла необхідність у якісно нових підходах до вирішення задач і проблем, які стоять перед нами. Але, незважаючи на всі ці вражаючі історії успіху, глибоке навчання все ще страждає від суворих обмежень. Якщо людська дитина може навчитися розпізнавати види тварин або клас предметів, бачачи лише кілька прикладів, для глибокої нейронної мережі зазвичай потрібно десять тисяч зображень для досягнення подібної точності. З іншого боку, сьогоднішні алгоритми, очевидно, далеко не розуміють сутність всередині сутності чи її класифікацію у тому розумінні, як це роблять люди. Багато прикладів показують, як навіть найсучасніші нейронні мережі вражають невдачі у випадках, які здаються людьми тривіальними. Одна із гіпотез, полягає в тому, що поточні обмеження в AI можна подолати лише завдяки алгоритмам нового покоління.

Завданням даної роботи є вивчення роботи поширених алгоритмів планування задач та знаходження альтернатив шляхом використання спеціалізованих структур та алгоритмів машинного навчання (прим. «машинне навчання» підрозділ науки, що вивчає штучний інтелект).

Для успішного виконання даної дипломної роботи необхідно:

- Опрацювати наукову літературу із обраної теми;
- Провести огляд існуючих алгоритмів планування, інструментів машинного навчання;
- Розробити алгоритм планування, який буде базуватися на принципах навчання AI;
- Створити модель програми для тестування розробленого алгоритму;

- Розробити програму із інтерфейсом користувача згідно прототипу.

Дана робота складається із чотирьох розділів. В процесі виконання завдання буде виконано огляд існуючих рішень, а також буде вибрано відповідну практичну базу, на основі якої буде виконано програмна реалізація планування на основі штучного інтелекту.

Перший розділ складається із огляду предметної області завдання. Тут будуть розглянуті базові поняття пов’язані із темою на яких буде грунтуватися теоретична складова даної роботи. В цей розділ буде включено опис уже розроблених методів й алгоритмів планування.

Другий розділ передбачає огляд застосовуваних технологій і алгоритмів для побудови тестової програми. Він також передбачає опис математичних моделей, аналізу ефективності алгоритмів машинного навчання, які стануть ключовими під час підготовки програмного забезпечення. Проведення вибору інструментів для реалізації програмного продукту.

Третій розділ передбачає опис розробки і реалізацію програми та інтерфейсу користувача за обраним макетом. Також тут буде наведений повний перелік кроків щодо установлення розробленого ПЗ на локальну машину для тестування.

Четвертий розділ включає в себе демонстрування результатів роботи моделювання розробленого алгоритму планування на основі штучного інтелекту, інтерфейс програми користувача, порівняльну характеристику по відношенню до інших алгоритмів.

В кінці роботи показано загальні висновки до всіх розділів, передбачено чому було остаточно вибрано використовувати розроблений алгоритм, та саме такі технології програмування, у порівнянні із іншими рішеннями у процесі розробки програмного забезпечення.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Якщо брати до уваги роботи Пінедо [9], то планування - це процес прийняття рішень, який застосовується до дуже широкого кола ситуацій теоретично і на практиці. Як результат, ресурси можуть охоплювати поняття від моделі операційних палат у лікарні або ядра в обчислювальній техніці. Аналогічно, завдання варіюються від хірургічних втручань в операційній до реальних процесів у комп'ютерній системі. Застосування практично безмежні. Як можна побачити, це дуже широке визначення.

На жаль, графік планування надзвичайно важко вирішити. Насправді, більшість проблем із плануванням належать до класу обчислювальних задач NP-класу [10]. Це також стосується більшості комбінаторних проблем оптимізації в багатьох інших галузях, таких як проблема розташування, маршрутизація, тощо. Однак можна стверджувати, що більшість реальних проблем планування є особливо важкими, і на практиці вони їх важче вирішити, ніж проблеми в інших сферах. Візьмемо для прикладу добре відому проблему маршрутизації транспортних засобів [11] про яку існує велика кількість написаної література. За даними досліджень та найпотужніших точних методів, нещодавно запропонованих для вирішення проблеми Балдачі у 2011, тільки у рамках 135 клієнтів та 7 транспортних засобів її можна вирішити оптимально. Для порівняння, для проблеми планування задач потрібно було чекати більше 20 років, щоб мати невеликі приклади із 10 задач та 10 машин, що оптимально вирішив Брукер у 1994 році, і не було досягнуто значних успіхів для оптимального вирішення прикладів із лише 20 задачами. Крім того, реальні проблеми планування набагато складніші, ніж звичайні описати ідеалізовані математичні моделі й вони мають багато побічних обмежень та узагальнень. Тому існує невелика сподівання на майбутнє вирішення даних питань [11][12][13].

Незважаючи на велику галузь проблем планування, у цій роботі було

вирішено зосережитися на конкретній проблемі, а саме на пошуку оптимальних рішень задачі статичного планування із обмеженими ресурсами. У противагу тому, щоб відповідати різним вимогам щодо варіацій проблем планування, єдиний алгоритм може надавати прийнятні результати для різних ситуацій загальної проблеми планування. Оскільки традиційні методи планування використовують правила пошуку або евристики, які характерні для конкретної моделі планування із сформулюваними обмеженнями, то нові методи можуть використовувати пряме подання графіків та алгоритмів пошуку й функціонувати без знання проблемного простору. У даний момент важливо поставити питання, чи потрібні оптимальнірішення на практиці чи ні. У найкращому випадку, моделі планування є просто хорошими наближеннями до реальних.

1.1 Проблема планування завдань із обмеженими ресурсами

1.1.1 Загальний огляд

Термін «планування» [14] стосується віднесення ресурсів до діяльностей (або діяльності до ресурсів) у конкретні моменти часу чи їх тривалості. Як вже було визначено - ці ресурси є обмеженими, і тому завдання повинні правильно ділитися, а точніше, конкурувати за них. Продуктивність алгоритму планування залежить від формулювання проблеми і можливості отримати користь від інформації, отриманої під час планування у вигляді збільшення продуктивності. Метою планування є знаходження такого графіка, який оптимізує заданий критерій або критерії.

У найбільш загальному вигляді проблема планування із обмеженими ресурсами визначається таким чином [2]:

- Сукупність діяльностей, які необхідно виконати.
- Сукупність ресурсів на яких будуть виконані діяльності.
- Сукупність обмежень, які повинні бути задовільнені.
- Супупність критеріїв, які будуть визначати продуктивність планування.

Для формалізації можна визначити загальну форму того що мається на увазі у процесі комплексного виконання попередніх пунктів [5]:

- Кожне завдання може бути виконано більш ніж одним способом, залежно від того, який ресурс(и) призначений для нього.
- Взаємозв'язки між задачами включають у себе можливіть накладення, тобто дана задача може почати оброблюватися у той момент, коли її предок частково виконався.
- Кожне завдання може бути перервано відповідно до заздалегідь визначеного набору режимів переривання (специфічного для кожного завдання), або переривання не допускається.
- Кожне завдання може потребувати більше одного ресурсу різних типів.
- Потреби в ресурсах завдання можуть відрізнятися протягом обробки завдання.
- Ресурси можуть бути відновлюваними (наприклад, робоча сила, машини) або не відновлювані (наприклад, сировина).
- Наявність ресурсів може змінюватись протягом тривалості планування чи завдання.
- Ресурси можуть мати тимчасові обмеження.

Для точного моделювання невизначеності, поширеної в реальних проблемах, загальне формулювання включає динамічні характеристики [2]:

- Доступність ресурсів може змінюватися.
- Вимоги до ресурсів можуть змінюватися.
- Цілі можуть змінюватися.

1.1.2 Поняття задачі

Завдання мають вимірювані оцінки таких критеріїв, як тривалість, вартість та витрачання ресурсів. Будь-яке завдання може потребувати одного ресурсу або набору ресурсів, і використання ресурсів може змінюватись протягом тривалості завдання. Оцінки тривалості та вартості можуть залежати від ресурса(ів), застосованих до завдання. Заходи щодо ефективності можуть бути

імовірнісними або детермінованими. Завдання може мати кілька режимів виконання. Будь-яке завдання може виконуватися більш ніж одним способом залежно від того, які ресурси використовуються для його виконання. Наприклад, якщо на роботу призначаються дві людини, то вона може бути виконана за половину необхідного часу, якщо б це зробила одна людина. Крім того, деталь може бути виготовлена за допомогою одного з трьох різних способів залежно від того, які верстати доступні. Одне завдання може складатися з декількох частин. Визначення частин включає специфікацію того, чи може завдання перериватися під час частин чи лише між частинами.

Крім того, деякі завдання можуть бути припинені в будь-який час, можливо, за певної вартості. Режими переривання можуть залежати від ресурсів, застосованих до завдання. Деякі завдання можуть бути перервані, але використовувані ними ресурси не можна використовувати в іншому місці, поки завдання не буде закінчено. Інші завдання можуть бути перервані, переназначені на інші ресурси, а потім будь-який ресурс повторно може бути застосований, коли завдання відновиться. Переміщення ресурсу з одного завдання на інше може бути можливим, як тільки було запущено перше завдання. Для інших завдань ресурс, щойно виконаний із завданням, повинен залишатися з цим завданням до його завершення. Деякі завдання можуть бути перервані, а потім перезапущені пізніше, але з деякою вартістю або погіршенням продуктивності або збільшенням прогнозованого часу до виконання. Завдання можуть використовувати ресурси в постійних або змінних кількостях протягом тривалості завдання. Можуть бути визначені завдання, які використовують певну комбінацію типів та типів ресурсів тощо.

1.1.3 Поняття ресурсу

Ресурси можуть бути поновлювані або не відновлювані [12][13][14]. Відновлювані ресурси доступні кожен період, не вичерпуючись. Приклади поновлюваних ресурсів включають робочу силу та багато видів обладнання.

Невідновлювані ресурси виснажуються в міру їх використання. Приклади невідновлюваних ресурсів включають капітал та сировину. Можна зауважити, що різниця між поновлюваними та невідновлюваними ресурсами може бути незначною. Ресурси залежать від можливостей, витрат та інших заходів щодо ефективності. Доступність ресурсу може відрізнятися. Ресурси можуть стати недоступними через непередбачені перебої, збої чи аварії.

Ресурси можуть мати додаткові обмеження. Багато ресурсів включають тимчасові обмеження, які обмежують періоди часу, коли вони можуть бути використані.

1.1.4 Обмеження й інші критерії

Обмеження та цілі визначаються під час постановки проблеми. Обмеження визначають доцільність планування. Цілі визначають оптимальність планування. Тоді як цілі мають бути досягнуті, обмеження повинні бути виконані. Як обмеження так і цілі можуть бути на основі завдань, на основі ресурсів, пов'язані з заходами ефективності або деякою їх комбінацією. Оптимальне планування не тільки задовольняє всі обмеження, але також є принаймні настільки ж хорошим, як і будь-яке інше можливе планування.

Проблеми планування проекта (project scheduling) [15] зазвичай визначають мінімізацію тривалості планування як основну мету. Однак більшість реальних проблем підпорядковані безлічі, часто суперечливих цілей. Часто цілі конфліктують між собою. Наприклад, можна просто скоротити тривалість планування, призначивши дорожчі ресурси для роботи над завданнями, але тоді вартість планування зростає. Оскільки більше цілей розглядається, то можливість конфліктів збільшується. Розгляд декількох цілей вимагає визначення механізму визначення взаємозв'язку між конфліктуючими цілями з метою прийняття рішень, які цілі є більш важливими.

Обмеження з'являються у багатьох формах. Обмеження прецеденту

визначає порядок виконання завдань. Тимчасові обмеження обмежують час, протягом якого можуть використовуватися ресурси та/або виконуватися завдання.

1.1.5 Project Scheduling

У задачах планування проектів [15] один проект складається з набору завдань або заходів (див. рис. 1.1). Завдання мають пріоритетні відносини, тобто деякі завдання не можуть бути запущені, поки їх попередники не будуть виконані. Завдання також мають орієнтовну тривалість і можуть включати різні інші характеристики, такі як вартість. Мабуть, найпоширенішою метою проблеми планування проекту є мінімізація часу на завершення проекту. Визначено багато спеціалізацій проблеми планування проекту. У проблемах планування проекту із обмеженими ресурсами завдання мають ресурси та ресурси є обмеженими. У проблемах планування проектів, що обмежені мультимодальними ресурсами, кожне завдання може бути виконане в більш

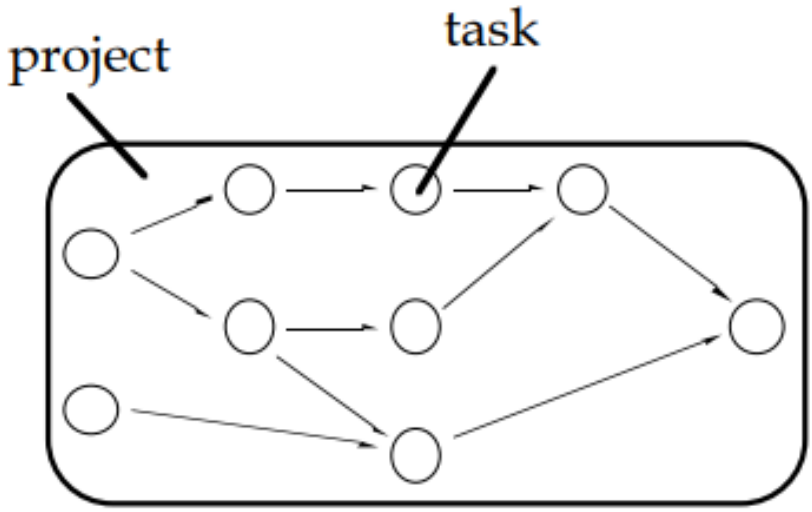


Рис. 1.1 - Project Scheduling [29]

ніж одному режимі, і кожен режим може мати різні вимоги до ресурсів. В проблемі із плануванням багатопроектних проектів, необхідно запланувати більше одного проекту.

Предметна область проекту [15] не є критичною, але вона дещо визначає складність проблем. Вона також визначає деталізацію визначення завдання та шкалу часу. Наприклад, у будівельному проекты, як правило, тривалість вимірюється місяцями, кожне завдання часто є складовою багатьох менших видів діяльності, а ресурси - підрядники та постачальники. З іншого боку, проект з розробки програмного забезпечення має часовий масштаб, вимірюваний тижнями, а ресурси - це особи, які працюють в команді розробників.

1.1.6 Job-Shop Scheduling

Типова проблема формулюється як робоче замовлення [16], яке складається із набору n завдань, кожне з яких містить m_i завдань. Кожне завдання має одного попередника і вимагає певного типу ресурсів. Перелік завдань можна призначити будь-якому з доступних ресурсів, але ресурс повинен бути правильного типу. До типових цілей можна віднести мінімізацію обсягу замовлення або строків виконання конкретних завдань чи завдання.

Можлива велика кількість варіантів як для планування роботи, так і для планування проекту. Деякі із цих варіантів включають розбиття на роботу,

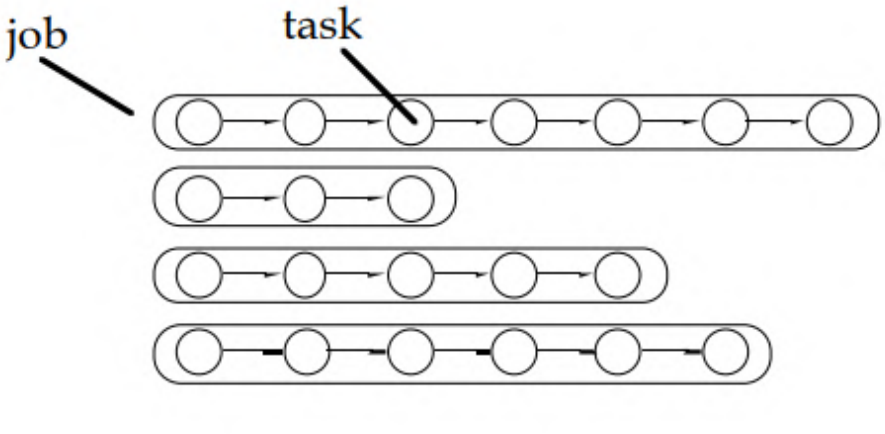


Рис. 1.2 - Job-Shop Scheduling [29]

попередження завдань, кілька режимів виконання, нерівномірну доступність та використання ресурсів, а також різні типи ресурсів.

1.2 Точні методи вирішення проблеми

Коли вперше були запропоновані рішення планування з обмеженими ресурсами, були використані прості моделі з точними методами вирішення проблеми. Враховуючи проблему, точні методи знаходять найкраще рішення (і гарантовано знайдуть найкраще рішення) щоразу, коли вони застосовані. Однак, оскільки були згодом додавалися певні обмеження, складність вирішення проблеми зростала, і просто знайти хороше рішення (або в деяких випадках можливе рішення) стало досить складною річчю. Крім того, багато методів займають занадто багато часу, коли застосовуються до проблем значного масштабу.

1.2.1 Метод критичного шляху

Метод критичного шляху (CPM) [17] забезпечує не обмежений ресурсом графік для набору операцій, які обмежені пріоритетом й з детермінованою тривалістю. Це дає найкоротший можливий простір, припускаючи наявність нескінченних ресурсів. Хоча цей метод є корисний для отримання грубого уявлення про складність виконання плану, метод критичного шляху не враховує тимчасові або ресурсні обмеження. Стохастичні варіації та динамічні зміни також були побудовані в спробі наблизити критичний шлях, що моделює припущення, яке є ближчим до реальності. Ці методи включають імовірнісні оцінки тривалості завдання.

1.2.2 Метод лінійного програмування

Багато проблем із плануванням можуть бути сформульовані у традиційній лінійній формі програмування, але лише за умови значних спрощень. Паттерсон [18] представив огляд оптимальних методів рішення для планування проекту. Як правило, точні методи залежать від характеристик цільової функції та конкретних формулювань обмежень. Як зазначав Лоуренс Девіс [4], багато обмежень, які часто зустрічаються в реальних задачах планування, не піддаються традиційним дослідженням операцій або

математичним програмуванням. Крім того, приклади методів лінійного програмування зазвичай не масштабуються, тому їх можна використовувати лише для конкретних випадків або невеликих проблем. Динамічний підхід до програмування був описаний Гелдом та Карпом [19], в якому оптимальний графік був поступово розроблений спочатку побудовою та оптимальним графіком для будь-яких двох завдань, а потім розширенням цього розкладу шляхом додавання завдань, поки не були заплановані всі завдання.

1.2.3 Метод дерева рішень

Багато методів рішення шукають дерево рішень [20], сформоване з пріоритетів у проектному плані. Як показано на рисунку 1.4, корінь дерева відповідає першому завданню. Другий рівень дерева - це набір завдань, які можна запланувати після того, як було заплановано перше завдання тощо. Таким чином, остаточне дерево являє собою здійсненний пріоритетом набір послідовностей завдань. Будь-яка з послідовностей кореня до листа може бути передана генератору розкладу. Крім того, послідовність завдань може бути запланована безпосередньо, якщо алгоритм генерації / обрізки дерева також враховує обмеження ресурсів. Пошук складається з обходу дерева до тих пір, поки не знайдеться найкращий шлях від кореня до листя. Чисельні методи, як правило, обмежуються за допомогою евристики, щоб зменшити розмір дерева. Неважко помітити, як швидко росте дерево при кількості заходів. Залежно від відносин пріоритету, кожне нове завдання може додати до дерева багато гілок. Коли завдання моделюються за допомогою декількох режимів виконання, кожен режим виконання додає ще один шар комбінаторних виборів для планувальника. Шпрехер і Дрексл [21] зазначають, що перелічувальні методи можуть вирішити проблему з багатьма різними цілями. Однак для зміни типів обмежень потрібен новий набір евристики для етапу планування або новий алгоритм обрізки для обрізки гілок дерев. Варіації методів розгалуження та зв'язаного рішення вперше були запропоновані в 1960-х роках.

Гілка та обмежений підхід Стінсона [22] генерували дерево шляхом планування діяльності, починаючи з першого завдання, а потім додаючи до дерева вузол для кожної задачі, яка може бути запланована на основі обмежень пріоритету та ресурсів. Межі на основі часткових графіків використовувались для обрізки дерева пошуку. Евристика для розширення дерева використовувала вектор із шести заходів. Як зазначають Шпрехер і Дрексл [21], перелічувальні методи не можуть вирішити великі проблеми; дерево просто занадто велике. Хоча було досягнуто значного прогресу в техніці обрізки, галузеві та зв'язані способи все ще обмежуються менш ніж сотнею

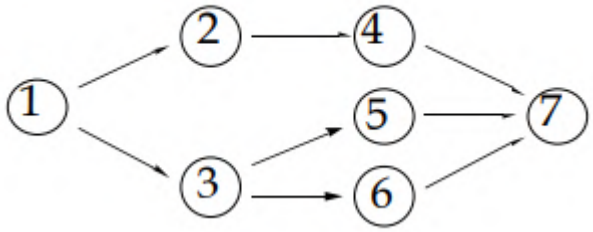


Рис. 1.3 [29]

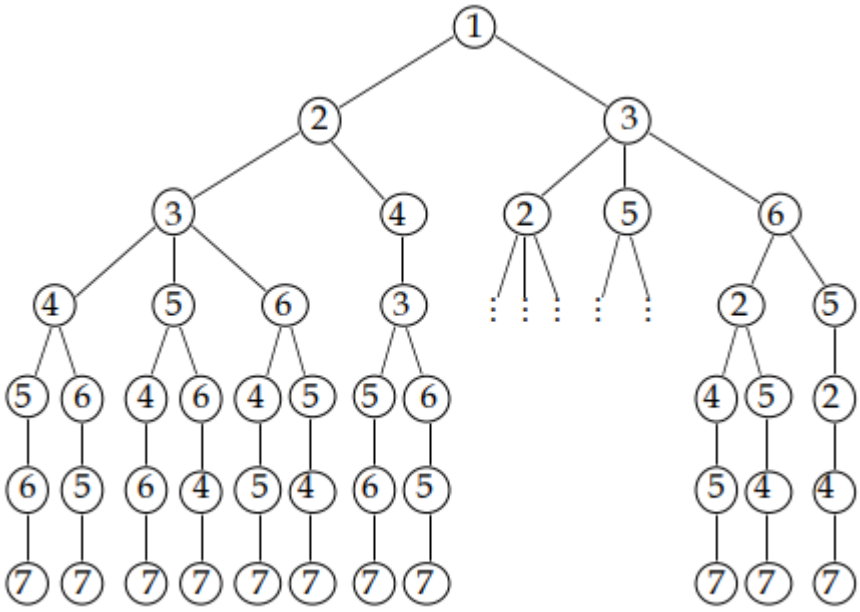


Рис. 1.4 – Метод обмеженого переліку [29]

задач або навіть меншою кількістю в мультимодальних випадках, і вони все ще потребують спеціальної евристики для урахування варіацій формулювання обмежень ресурсів.

1.3 Евристичні методи вирішення

Тоді як точні методи рішення гарантовано знайдуть оптимальне рішення (якщо таке існує), евристичні методи іноді знаходять оптимальні рішення [2], але частіше знаходять просто «хороші» рішення. Евристичні методи зазвичай вимагають значно менше часу та/або місця, ніж точні методи. Евристика вказує, як прийняти рішення за певної ситуації; евристика - це правила для вирішення, яку дію вжити. Евристику при плануванні часто називають правилами планування або правилами відправки. Визначення цих правил часто досить складне, і більшість з них пристосовані до конкретного типу проблеми з дуже специфічним набором обмежень та припущень. Евристика може бути детермінованою - коли щоразу отримують однаковий результат - або може бути стохастичною - при кожному запуску вона може давати інший результат. Вона може виконувати одне правило за один раз, або вона може бути здатною до паралельних рішень. Гібридні алгоритми можуть поєднувати багато евристик. Традиційні евристичні методи зазвичай виконуються у три етапи: проектування графіку, послідовність дій, потім планування. Одні методи використовують евристику для пошуку комбінаторного простору перестановок у послідовностях завдань, інші використовують евристику для визначення можливих присвоєнь часу/завдання/ресурсів під час генерації розкладу, а інші використовують евристику для поєднання послідовності дій і планування. Гібридні рішення намагаються підтримувати більше ніж одне представлення або поєднувати кілька методів пошуку або алгоритмів, які задовільняють задані обмеження.

1.3.1 Евристика планування

Евристика планування діє на наборі завдань і визначає, коли кожне завдання слід виконати. Якщо завдання може виконуватися в більш ніж одному режимі виконання або на будь-якому з наборів ресурсів, евристика також повинна визначити, які ресурси та/або режим виконання використовувати.

Загальна список евристики наведений в таблиці 1. Планувальник здійснює умови обмеження, призначаючи завдання ресурсу (або ресурсу завдання) в той момент, коли ресурс доступний і завдання може бути виконано.

Таблиця 1.1 Список евристик

Евристика	Що вона робить
MIN SLK	вибирає завдання з найменшою загальною неактивністю
MIN LFT	вибирає завдання з найближчим часом закінчення
SFM	вибирає режим виконання з найменшою можливою тривалістю
LRP	вибрати режим виконання з найменшою часткою ресурсів

1.3.2 Евристика послідовності

Тоді як евристика планування працює над завданнями, які вирішують, коли вони повинні бути виконані, евристика послідовності визначає порядок, у якому завдання будуть заплановані. Ці евристики часто використовуються в поєднанні з деревами рішень, щоб визначити, яку частину дерева шукати або уникати. Наприклад, обмежений пошук розбіжностей із зворотним відстеженням використав Вільям Гарві та Метью Гінсберг [23] для дуже ефективного вирішення деяких класів задач планування, коли послідовність для завдань планування структурується як дерево рішень.

1.4 Методи вирішення пов’язані із штучним інтелектом

Гілдум згрупував підходи штучного інтелекту до планування як експертної системи, так як і на основі знань. Обидва є структурованими евристичними методами, які відрізняються за способом контролю за застосуванням своєї евристики, що є специфічною для застосування.

Експертні системи складаються із бази правил, знімка поточного рішення та ядра висновків. Двигун умовиводу визначає, як правила if-then в базі правил застосовуються до поточного рішення для здійснення пошуку. База правил може бути розширена в міру просування рішення. База правил

підходить явно до конкретної проблеми, тому експертні системи, як правило, є високоспеціалізованими.

Системи, що базуються на знаннях, зазвичай розбивають проблему на додаткові проблеми або різні аспекти. Визначаються "агенти", кожен з яких стосується певного аспекту рішення. Кожен агент спрямовує рішення у напрямку, що найбільше має відношення до цього агента. Варіанти алгоритмів включають комбінації мікро- та макро модифікацій рішень, а також типи атрибутів, на які агенти налаштовані відповідати. Гілдум розрізняв три загальновідомі рішення штучного інтелекту:

- ISIS (Intelligent Scheduling and Information System)
- OPIS (Opportunistic Intelligent Scheduler)
- MicroBOSS (Micro-Bottleneck Scheduling System)

Виходячи з правил, якими вони керували в пошуку. Хільдум зазначив, що його власний метод, DSS (система динамічного планування), в основному є множинним атрибутом, динамічним евристичним підходом, який фокусується на найбільш нагальній невирішеній проблемі в будь-який момент часу.

1.4.1 Метод імітації відпалу

Методи відпалу дозволяє перейти до гірших рішень і, таким чином, часто уникає місцевих неоптимальних рішень. Аартс, Лаарховен та Ленстра описали [1] один із перших модельованих підходів відпалу до задач планування.

Боктор повідомив про досить хороші показники завдяки симульованому підходу імітації відпалу щодо проблем Паттерсона. У цій роботі [2] модельований відпал був використаний для пошуку комбінаторного простору перестановок послідовностей. Враховуючи послідовність завдань породжених підпальником, ця евристика може бути використана для створення планування. Цей метод прямо пов’язаний із методом дерева, але імітований відпал може бути застосований до значно більших розмірів проблем. За більш як 20 років користуванням даним алгоритмом приходять до наступних висновків [3]:

- Тільки велика кількість обчислень забезпечує якісний розв’язок.
- Даний алгоритм має великі плюси через свою загальність застосування й простоту реалізації.
- Застосовується у багатьох практичних ситуаціях, де наявні алгортми не працюють.

1.4.2 Еволюційні алгоритми

Один з найбільш ранніх запропонованих варіантів використання генетичних алгоритмів для планування було здійснено Лоуренсом Девісом. У своїй роботі [4] Девіс відзначив привабливість використання методу стохастичного пошуку через розмір простору пошуку та запропонував непряме подання, в якому генетичний алгоритм діяв за списком, який потім був розшифрований для формування фактичного розкладу. Зокрема, він зазначив важливість збереження виконуваності у представництві. Девіс зауважив [4], що багато реальних проблем планування включають в себе шари неправильно визначених обмежень, які часто важко, якщо не неможливо уявити, використовуючи традиційні методи математичного програмування. Відзначаючи, що рішення, засновані на знаннях, як правило, детерміновані і, таким чином, сприйнятливі до входження в неоптимальні області пошукового простору, Девіс висловив припущення, що генетичні алгоритми, в силу своєї стохастичної природи, уникатимуть неоптимальних рішень. Починаючи з документа Девіса [4], запропоновано численні реалізації не тільки для проблеми планування робочих місць, але й інших варіантів загальної проблеми планування із обмеженими ресурсами. В деяких випадках представлення для одного класу проблем може застосовуватися і до інших. Але в більшості випадків модифікація визначень обмежень вимагає іншого представлення.

Ральф Брунс [5] узагальнив підходи до планування виробництва за чотирма категоріями, що перекриваються: прямі, непрямі, незалежні від доменної моделі та конкретні уявлення. Більшість підходів генетичного

алгоритму використовували непряме уявлення.

Ці методи характеризувались традиційними бінарними поданнями або представленнями на основі порядку. Інформація, що стосується проблеми, використовувалася в деяких непрямих методах для підвищення продуктивності, але це все ще були методи на основі списку чи порядку, вони вимагали перетворення з геному в графік, а в деяких випадках вимагали також побудови графіка. Відзначаючи зворотну залежність між загальністю алгоритму та його продуктивністю, представлення Брунса [5] було налаштоване "на максимально ефективну роботу з проблемою планування виробництва". Це пряме, специфічне для конкретного завдання представлення використовувало перелік призначень замовлень, у яких послідовність замовлень не була важливою.

Багчі порівняв [6] три різних подання і зробив висновок, що чим більше інформація, що стосується конкретної проблеми, буде включена в представлення, тим краще буде працювати алгоритм.

Пізніше Філіп Хасбенд [7] окреслив найсучасніші генетичні алгоритми планування. Хасбенд відзначив схожість між плануванням і проблемами, які базуються на таких задачах як проблема комівояжера. Він також посилався на інші важкі проблеми з NP, такі як проблеми з компонуванням та упаковкою у рюкзак тощо. У наборі попередніх тестів з різними уявленнями автор спробував послідовно-графічний підхід для планування проекту, подібний до модельованого підходу відпалу Боктора [2] але з генетичним алгоритмом. Представництво являло собою гібрид на основі послідовностей; генетичний алгоритм генерував послідовності, а потім послідовності були евристично заплановані. Результати не були обнадійливими; тільки мутація (тобто випадковий пошук) виконується краще, ніж генетичний алгоритм.

Ця ефективність є аналогічною різниці в ефективності між прийняттям порогу та генетичним алгоритмом для проблеми комівояжера. Симульований підхід відпалення Боктора виконується краще, ніж генетичний алгоритм, в першу чергу завдяки кращій евристиці планування (Боктор реалізував паралельний планувальник [4], який «заглядує» у майбутнє). Інший фактор був метод упорядкування. У реалізації генетичного алгоритму автор використовував кросовер з частковою відповідністю, тоді як Боктор робив випадкове перемикання на основі пріоритету. Як ілюструє різниця між рекомбінацією краю та частковою відповідністю проблеми комівояжера, вибір оператора кросовера може означати різницю між генетичним алгоритмом, який працює, і таким, який не відповідає поставленій задачі. Більш висока продуктивність можлива за допомогою генетичного алгоритму, але тільки з кросоверним оператором, пристосованим до проблеми.

Таблиця 1.2. Порівняння алгоритмів оптимізації

	Генетичний алгоритм	Метод імітації відпалу
Пам’ять	- Потребує наперед заданої кількості пам’яті. - Використовується для зберігання й генерування нових популяцій.	- Майже не використовує пам’ять. - Використовується для рестарту пошуку після кожної редукції температури.
Вибір даних	- Безсистемний спосіб - Базується на невеликій частині існуючого рішення, що залежить від сфери застосування.	- Пошук незначний, оскільки він прогресує із меншими змінами прийняття різних рішень по відношенні до поточного.

Продовження таблиці 1.2

	Генетичний алгоритм	Метод імітації відпалу
Варіація	- Нові простори рішення за рахунок поєднання вже існуючих рішень. - Наявність великої кількості варіацій через генерацію нових випадкових популяцій рішення.	- Кожна варіації імітується відповідно до фізичного процесу відпалу. - На ранніх стадіях пошуку шанси прийняти рішення, що значно знижують продуктивність, вищі.
Сфера застосування	- Локальна оптимізація проблеми.	- Глобальна оптимізація проблеми.
Швидкодія	- За короткий проміжок часу надає ціле сімейство рішень.	- Потрібні значні обчислювальні ресурси для рішення.

Жоден з еволюційних алгоритмів не може визначити, чи графік є неможливий. Проблеми, для яких були розроблені ці методи, мали можливі рішення, але в реальних проблемах здійсненність не гарантується. Наприклад, у проблемі планування проекту, якщо всі ресурси доступні в постійній кількості, можливий графік завжди можна знайти, просто продовживши тривалість проекту, поки всі ресурси не будуть обмежені. Коли ресурси є доступні лише через певні проміжки часу, такої гарантії доцільності не існує.

Висновок до розділу 1

В результаті проведених досліджень, направлених на виконання аналізу проблеми ефективного планування, огляду його типів у обчислювальних системах та збір даних про поширені алгоритми планування можна прийти до наступних висновків:

1. Складання графіку планування є нетривіальною задачею. Часто приходиться мати справою із задачею передбачення, оскільки для даного класу проблем часто неіснує оптимального рішення за прийнятний час.
2. Розглянуто методи щодо вирішення проблеми планування із обмеженими ресурсами, що стосуються галузі штучного інтелекту. Зокрема порівняно метод «імітації підпалу» та сімейство генечних алгоритмів.
3. Було вирішено вибрати саме генетичний алгоритм як базовий підхід щодо реалізації планування на основі штучного інтелекту, оскільки поставлена задача вимагає того, щоб прийнятний графік планування був готовий у найкоротші строки, чого метод «імітації відпалу» забезпечити не може (див. табл. 1.2).

Як результат було вирішено звузити область даної роботи й зосередитися на розробці ефективного планувальника для систем реального часу, алгоритм якого буде базуватися на методиках й підходах GP.

РОЗДІЛ 2. ВИБІР І ОБГРОНТУВАННЯ ЗАСОБІВ РЕАЛІЗАЦІЇ

Перед тим як перейти до проектування й розробки програмного продукту необхідно описати базові концепції, вибрати архітектуру й програмні й апаратні засоби, що будуть необхідні у подальшому. Метою цього розділу є обгронтування засобі реалізації майбутньої моделі планування на основі штучного інтелекту.

2.1 Постановка математичної моделі планування

Для ефективної розробики алгоритма планування на основі штучного інтелекту спершу необхідно розглянути математичну модель планування [15] із якою прийдеться мати справу. Для цього коротко оглянемо основні моменти й понятння.

Більшість моделей планування визначаються множиною J, яка складається з n робіт із певними послідовними індексами: $J = \{J_1, J_2 \ldots J_n\}$ Аналогічно ресурси, які моделюються представлені множиною M, відповідно, що $M = \{M_1, M_2 \ldots M_m\}$, де m – кількість наявних машин (ресурсів) у системі. Ці множини вважаються незалежними один від одної. Індкси j і k використовуються для позначення робіт, а і та l відповідно – машин. Планування може бути представлене діаграмами Гранта. Діаграми Гранта можуть бути орієнтованими на машину (рис. 2.1) або орієнтовані на роботу (рис. 2.2).

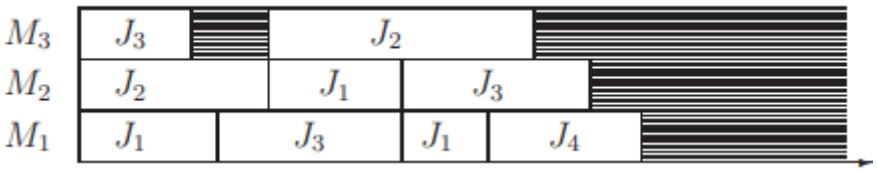


Рис. 2.1 – Машинно-орієнтована діаграма Гранта [31]

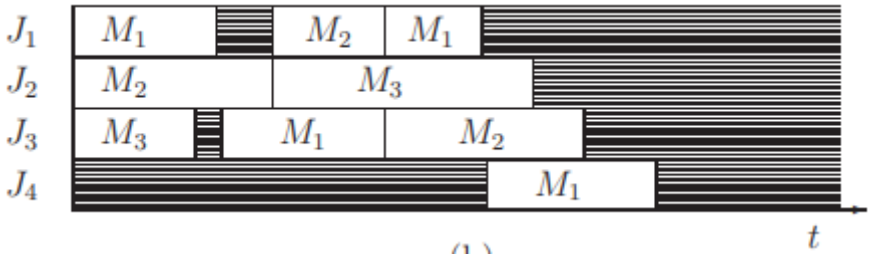


Рис. 2.2 – Робото-орієнтована діаграма Гранта [31]

Основні визначення, що пов’язані процесом розподілення робіт на ресурси [6]:

- Маршрут обробки. Для заданої роботи j маршрутом є впорядкований список ресурсів, що повинні бути відвідані для успішного виконання роботи. Якщо робота повинна відвідати три машини (наприклад, 2, 1 і 6), то це означає, що вона має три завдання. Зазвичай і, слідуючи цьому невеликому прикладу, завдання в машині 1 не може початись, поки попереднє завдання тої ж роботи на машині 2 не закінчиться.
- Часи обробки. Зазвичай позначається як p_j або p_{ij}. Це кількість часу потрібного на те, щоб дана робота j була виконана на машині і. Цей час фіксований, і зазвичай він не може бути перерваним. Машина і зайнята під час обробки завдання j і не може бути зайнята протягом цього часу будь-якою іншою роботою.
- Термін виконання конкретної роботи та вага. Вони позначаються відповідно d_j і w_j. У встановлений термін визначається час, коли робота j повинна бути виконана на у системі. Вага фіксує відносну важливість роботи. Ці дані будуть використані пізніше для досягнення цілей оптимізації.
- C_j – це модель часу, коли останнє завдання виконується на останній машині.
- L_j – є мірою затримки часу виконання роботи j щодо його строку d_j. $L_j = C_j - d_j$.

- T_j – міра затримки роботи j, де $T_j = \max\{L_j, 0\}$.

Звичайно, для вирішення проблеми розподілу ресурсів й задач може знадобитися велика кількість додаткових даних, залежно від реальної проблеми, яку потрібно вирішити. Однак і для стислості обмежемося попередніми даними.

Із вищезазначених термінів, можна виділити два напрямки оптимізації роботи планування:

- Базуючись на часі виконання. Найбільш поширеним є мінімізація максимального часу закінчення робіт, а саме,
- $C_{max} = \max\{C_1, C_2, \dots C_n\}$. Це питання в основному пов'язане із максимальним збільшенням утилізація машини. Друга об'єкт вивчення - це мінімізація загального часу роботи, що визначається як $\sum C = \sum_{j=1}^{n} C_j$.
- Базуючись на термінах виконання конкретної роботи. Найбільш поширеною є мінімізація загальної затримки наявних робіт, а саме, $\sum T = \sum_{j=1}^{n} T_j$. Тим не менш, версія формули, яка включає у себе ваговий коефіцієнт w, або $\sum wT = \sum_{j=1}^{n} w_j T_j$ несе більш реалістичний характер.

2.2 Опис алгоритма штучного інтелекту

Методи планування, які засновані на евристиці, що були розглянуті у попередньому розділі мають суттєві недоліки. У одних випадках вхідних даних вони дають хороші результати, а для деяких інших – значно гірші.

Як одне із рішень проблема планування – це генетичні алгоритми (GA), які є засновані на механіці природного відбору та природньої генетики [10]. Основна мета, що стоїть за дослідженнями генетичних алгоритмів – це надійність. Генетичні алгоритми є одними із найпопулярнішими методи випадкового пошуку для різних видів проблеми планування завдань.

2.2.1 Огляд основних термінів й понять

Генетичний алгоритм постійно намагається вдосконалити середній показник «придатності» (або fitness у зарубіжній літературі) популяції за рахунок створення нових популяції. Якість рішення значною мірою залежить від вибору деяких ключових параметрів, таких як функція fitness, розмір популяції, ймовірність схрещування та ймовірність мутації. Деякі ключові терміни (див. рис.2.3), які використовуються в GA:

- Ген: елементарна частинка, яка представляє частину загального рішення.
- Хромосома: це набір генів, що представляють власне саме рішення.
- Популяція: це кількість хромосом, які доступні для тестування.
- Локус: унікальна позиція, яку може зайняти ген у хромосомі.

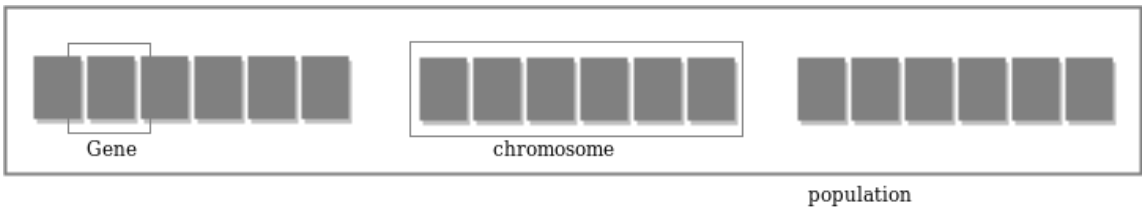


Рис. 2.3 – Відношення між термінами [30]

Генетичний алгоритм використовує три оператори, відомі як природні відбір, схрещування та мутація. Генетичний алгоритм відрізняється від традиційних методів оптимізації у наступному:

- Генетичні алгоритми є одночасно й надійними, й ефективними.
- Вони можуть виробляти високооптимальні рішення.
- Здатність використовувати виокремлені характеристики попередніх спроб розв’язання проблеми на основі яких можуть бути побудовані більш кращі рішення.
- Легкі у обчислюваннях і прості у реалізації.
- Генетичні алгоритми базуються на законах теорії ймовірності, а не на детермінованих правилах.

2.2.2 Структура генетичного алгоритму

Структура генетичного алгоритму для будь-якої проблеми залежить від п’яти речей:

- Вибір представлення хромосом.
- Побудова генетичних операторів.
- Вибір функції fitness.
- Ймовірності, які можуть контролювати виконання генетичних операторів.
- Кількість популяцій.

Кожна із вищеописаних кроків сильно впливає на отримані рішення, а також продуктивність генетичного алгоритму. Зазвичай структура генетичного алгоритму (див. рис. 2.4) складається із наступних кроків:

- Крок 1: Ініціалізація – ініціалізація популяції.
- Крок 2: Оцінювання – оціниті кожну хромосому, використовуючи fitness-функцію.
- Крок 3: Етап виконання генетичних операцій – вибрати хромосому і застосувати генетичні оператори на ній, для вироблення нової хромосоми (популяції).
- Крок 4: Повторити кроки 2 та 3, поки не буде досягнуто умови припинення.

З вищенаведених кроків ми бачимо, що генетичні алгоритми використовують концепцію виживання найбільш «пристосованих», передаючи «хороші» хромосоми до наступного покоління, і поєднуючи різніпопуляції для пошуку нових результатів.

2.2.3 Ініціалізація популяції

Проектування структури хромосом має вирішальне значення для розробки генетичного алгоритму. Ми визначаємо нашу структуру хромосом як

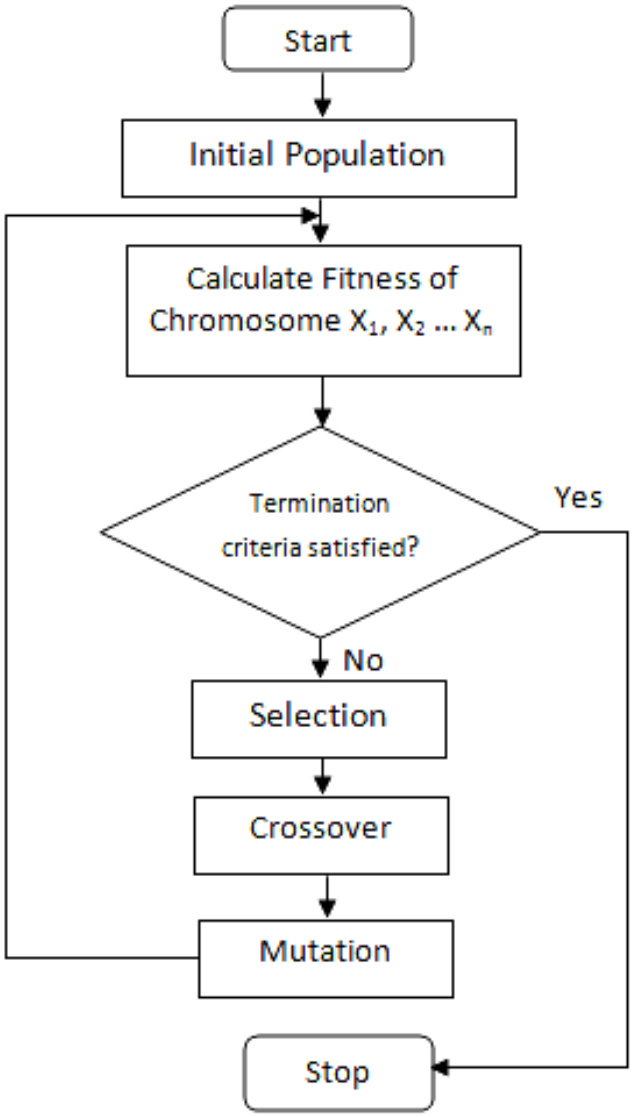


Рис. 2.4 – Структура генетичного алгоритму [32]

комбінацію двох понять - SQ і SP, довжина яких дорівнює кількості завдань. Черга планування (Scheduling queue, далі SQ) описує порядок черги завдань, а запис у SQ являє собою завдання, яке повинно бути розплановане відповідно до графіку планування. Запис у процесор планування (Scheduling processor, далі SP) представляє собою відповідне завдання, яке буде заплановане на ньому. Щодо деталей створення хромосоми можна виділити такі кроки:

- Крок 1: Вибрати випадковим чином одне завдання із списку усіх завдань. Поставити це завдання як перше завдання в SQ.
- Повторити крок 3 протягом (v-1) разів.

- Крок 3: Випадково вибрати завдання, якого немає в SQ і у якого всі предки бути у SQ й додати це завдання у чергу.
- Для частини, що стосується SP, випадково згенерувати ціле число у діапазоні між 1 і m для кожного завдання і додати це у SP.

2.3.4 Оцінка fitness функції та природній відбір

Fitness-функція в генетичних алгоритмах зазвичай є цільової функцією, яку ми хочемо оптимізувати в задачі. Вона використовується для оцінки хромосом. Для розрахунку fitness-значення та для того, щоб відібрати хороші хромосоми, ми визначити fitness-функцію як:

$$F\ (i)\ =\ (maxFT - FT\ (i) + \frac{1}{(maxFT - minFT + 1)} \qquad (2.1)$$

де, maxFT і minFT – це максимальний і мінімальний час закінчення хромосом в поточній популяції, відповідно. $FT\ (i)$ – час закінчення і-ї хромосоми.

Після того, як оцінюються fitness-значення всіх хромосом, ми можемо вибрати хромосоми з більш високим значенням fitness за допомогою механізму «колесо рулетки» (див. рис. 2.5). Кожна хромосома в популяції займає розмір сектору, пропорційний значенню fitness-функції. Випадкові цілі числа генеруються і використовуються як індекс у колесі рулетки, щоб визначити, яка хромосома буде передана наступному поколінню. Оскільки хромосоми з більш високим значенням fitness матимуть більші площі секторів, вони, швидше за все, будуть відібрані та передані наступному поколінню.

2.2.5 Репродукція: Схрещування та мутація

Схрещування – це механізм, який виробляє нове потомство, яке містить деякі частини генетичного матеріалу обох предків. Як правило для спрощення й пришвидшення процедури схрещування використовують бітове представлення деякої характеристики гену.

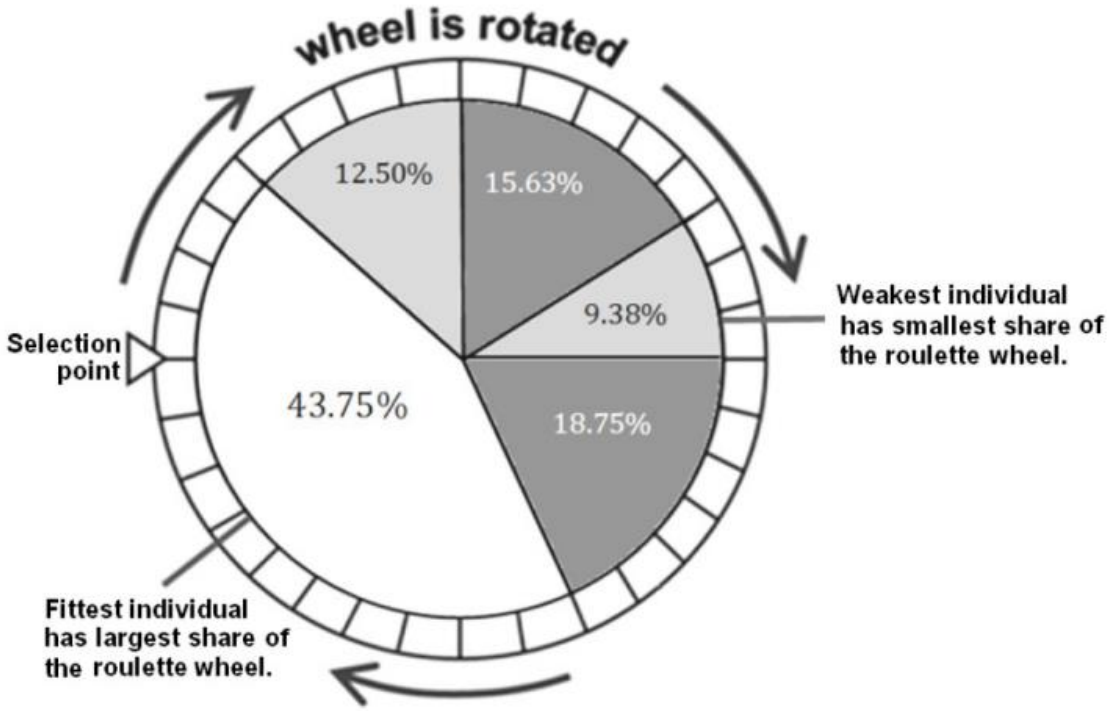


Рис. 2.5 – Ілюстрація принципу «Колесо рулетки» [33]

Існують такі типи схрещування (див. Рис. 2.6):

- Відносно однієї точки – вибирається єдина точка до якої біти залишаються незмінними, а після – змінюються на відповідні батьківські.
- Відносно двох точок – аналогічно до попереднього, але заміні підлягає тільки діапазон бітів впередині, а решта бітів залишаються попередніми.
- Рівномірний – кожен біт розглядається окремо й у нових хромосомах утворюється за наперед визначеним правилом.

Оскільки наша хромосома складається з двох окремих частин SP і SQ, що мають різні характеристики, для кожної частини ми застосовуємо різні алгоритми схрещування. Ми випадковим чином вибираємо одну або другу частину хромосоми і застосовуємо для цих двох частин дві різні операції схрещування.

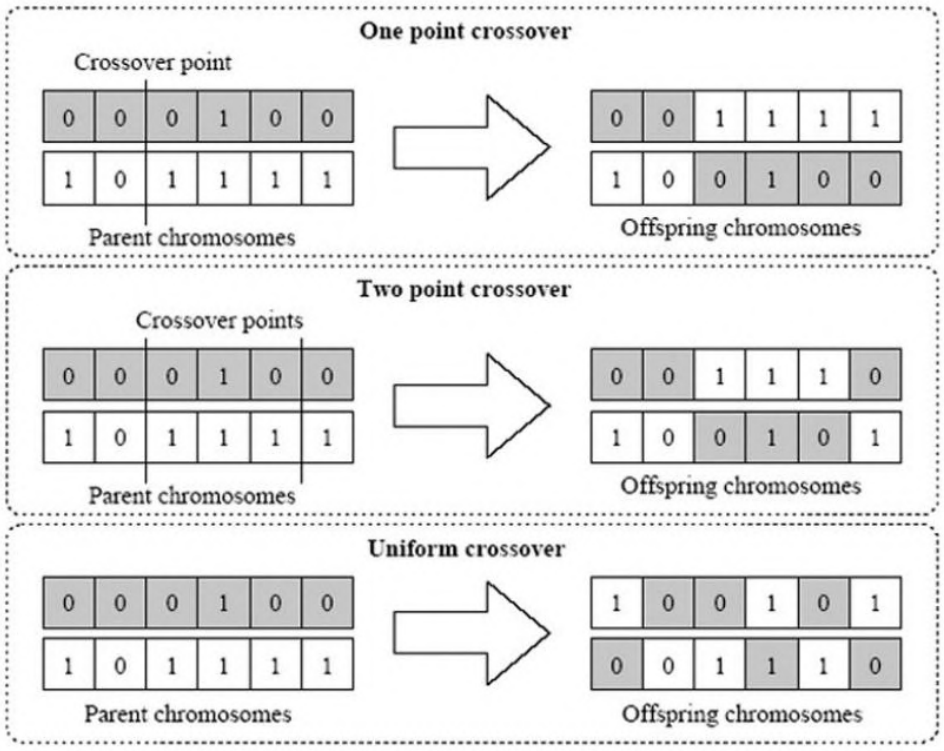


Рис. 2.6 – Типи схрещування [34]

Послідовність алгоритму схрещування складається із таких кроків:

- Крок 1: Задати ймовірність схрещування – P_c.
- Крок 2: Випадково вибрати пари хромосом і згенерувати число $pNum$ з плаваючою комою у діапазоні між 0 і 1 для кожної такої пари.
- Крок 3: Якщо $pNum \leq P_c$, то повторити крок 4 і крок 5.

  Інакше: безпосередньо додати ці дві хромосоми у наступну популяцію.
- Крок 4: Випадковим чином згенерувати дві точки, q та p у діапазоні від 1 до v і біт схрещування CF.
- Крок 5: Якщо $CF = 0$, то змінити порядок завдань у SQ між p і q однієї хромосоми відповідно до порядку завдань іншої хромосоми, залишок хромосом зберігається без змін.

  Інакше: Провести обмін частиною в SP між p і q двох хромосом, а решту двох хромосом залишити без змін

Мутація – генетичний оператор [11], який змінює одне або кілька значень гена в хромосомі по відношенню до початкового стану (див. рис. 2.7).

Це може призвести до додавання до популяції абсолютно нових хромосом. За допомогою цих нових хромосом генетичний алгоритм може досягти кращого рішення, ніж це було раніше можливо. Мутація може розглядатися як випадкова зміна особистості.

Опишемо наступний алгоритм у застосуванні мутації для вирішення завдання:

- Крок 1: Ввести ймовірність мутації – P_m.
- Крок 2: Для кожної хромосоми згенерувати число pNum з плаваючою комою у діапазоні між 0 і 1.
- Крок 3: Якщо $pNum \leq P_c$, то повторити крок 4 і крок 5.
  Інакше: безпосередньо додати хромосому у наступну популяцію.
- Крок 4: : Випадковим чином згенерувати точку p у діапазоні від 1 до v і біт мутації MF.
- Крок 5: Якщо MF = 0, то процесор залишається таким же як у черзі SP.
  В іншому випадку довільно змінюють процесор у діапазоні між 1 і m із SP.

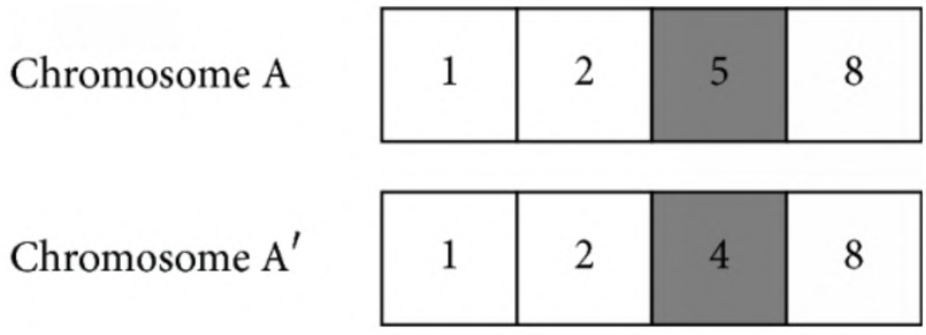


Рис. 2.7 – Приклад мутації [35]

2.3 Вибір засобів та середовища розробки

Перед тим як перейти до власне реалізації моделі планування на основі штучного інтелекту необхідно коротко визначитися із середовищем майбутньої роботи програми**.**

2.3.1 Вибір засобів розробки

По-перше, необхідно вибрати конкретну мову програмування. Оскільки дана робота безпосередньо пов’язана із AI, то доцільно було б вибрати саме таку мову, яка є актуальною у даній області. У інтернеті є багато різноманітних тематичний рейтингів пов’язаних із даним питання. Розглянемо декілька найпоширеніших мов програмування для написання програм у області штучного інтелекту базуючись на статистиці [24].

a) Python

Python вважається на першому місці в списку всіх мов програмування пов’язаних із AI через свою простоту. Синтаксис, що належать до Python, дуже простий і його можна легко вивчити. Тому в ньому можна легко реалізувати багато алгоритмів AI. Розробка на Python займає короткий час порівняно з іншими мовами, такими як Java, C або Ruby. Python підтримує об’єктно-орієнтований, функціональний, а також процедурно-орієнтований стилі програмування. У Python є багато бібліотек, які полегшують завдання. Наприклад: Numpy – це бібліотека Python, яка допомагає вирішувати багато наукових обчислень. Також у Python є бібліотека Pybrain, яка призначена для використання машинного навчання в Python.

b) R

R – одна з найефективніших мов та середовищ для аналізу та обробки даних для статистичних цілей. Використовуючи R, можна легко створити хорошої якості графіки, включаючи математичні символи та формули, де це необхідно. Крім мова загального призначення, R має численні пакети, такі як RODBC, Gmodels, Class та Tm, які використовуються в галузі машинного навчання. Ці пакети полегшують реалізацію алгоритмів машинного навчання для вирішення проблем, пов’язаних з бізнесом.

c) Lisp

Lisp – одна з найдавніших і найбільш підходящих мов для розвитку в AI. Вона була винайдена Джоном Маккарті, батьком штучного інтелекту в 1958 році. Lisp має можливість ефективно обробляти символічну інформацію. Він також відомий своїми чудовими можливостями прототипування та легким динамічним створенням нових об’єктів, з автоматичним збиранням сміття. Її цикл розробки дозволяє інтерактивно оцінювати вирази та рекомпіляцію функцій чи файлів під час роботи програми. За роки, завдяки просуванню, багато з цих особливостей перенеслись на багато інших мов, тим самим впливаючи на унікальність Lisp.

d) Prolog

Ця мова є поряд з Lisp, коли ми говоримо про розвиток в галузі AI. Особливості, що надаються нею, включають ефективне узгодження шаблонів, структурування даних на основі дерева та автоматичне зворотнє відстеження. Всі ці функції забезпечують напрочуд потужні та гнучкі підходи програмування. Prolog широко застосовується для роботи над медичними проектами, а також для проектування експертних систем штучного інтелекту.

e) Java

Java також може розглядатися як хороший вибір для розробки AI. Галузь штучного інтелекту включає у себе алгоритми пошуку, штучні нейронні мережамі та генетичне програмуванням. Java надає безліч переваг: просте використання, простота налагодження, пакетні послуги, спрощена робота з масштабними проектами, графічне представлення даних та краща взаємодія з користувачами. Вона також включає в себе Swing і SWT (Стандартний інструментарій віджетів). Ці інструменти роблять графіку та інтерфейси привабливими та витонченими.

Аналізуючи вищенаведену інформацію було вирішено зупинитися на мові програмування Python. Вона задовільняє всім вимогам й до того ж важливим фактором вибору стало саме її лідерство у рейтингу. Також важливими показниками стали такі фактори як:

- Простота розробки.
- Швидкість розробки.
- Широка підтримка зі сторони розробників (комп'юніті).
- Велика кількість відкритих ресурсів, які вирішують багато практичних задач.

2.3.2 Вибір середовища розробки

Оскільки вибір мови програмування вже зроблено, необхідно вибрати власне середовище розробки. Python – це надзвичайно популярна платформа для розробки програмного забезпечення. Тому не дивно, що існує велика кількість наборів засобів, що спрощують розробку на цій мові [25]. Першим фільтром, який буде відігравати значну роль у даному виборі є те, щоб вибране середовище було безкоштовним. Додатковими вимогами будуть простий інтерфейс й поширеність серед розробників даної платформи.

a) Sublime Text 3

Sublime Text - поширений редактор коду, який реалізує підтримку багатьох мов, включаючи Python. Він має величезне співтовариство, швидкий, легко налаштовується.

Він має базову вбудовану підтримку Python при його встановленні. Однак ви можете встановити такі пакети, як налагодження, автоматичне доповнення, зв'язування коду тощо. Також тут є різні пакети для наукових розробок, Django, Flask тощо. В основному можна налаштувати Sublime текст для створення повноцінного середовища розробки Python відповідно до ваших потреб. Можна завантажувати та використовувати програму протягом невизначеного періоду часу. Однак час від часу отримується спливаюче вікно

із зазначенням, що "вам потрібно придбати ліцензію для подальшого використання".

b) Atom

Atom - це редактор коду з відкритим кодом, розроблений Github, який можна використовувати для розробки Python (подібний текст Sublime). Його функції також схожі на Sublime Text. Atom дуже різноманітно настроюється. Ви можете встановити пакети відповідно до ваших потреб. Деякі з найбільш часто використовуваних пакетів в Atom для розробки Python - це автозаповнення-python, linter-flake8, python-debugger тощо. Багато розробників віддають перевагу Atom перед Sublime Text для розробки на Python.

c) Pycharm

PyCharm - це IDE для професійних розробників. Вона створена компанією JetBrains, яка відома у створенні чудових інструментів розробки програмного забезпечення. Існує дві версії PyCharm:

- Community - безкоштовна версія з відкритим кодом, легка, хороша для Python та наукових розробок.
- Professional - платна версія, повнофункціональний IDE з підтримкою веб-розробки.

PyCharm надає всі основні функції, які повинен забезпечити хороший IDE: завершення коду, перевірка коду, виділення помилок та виправлення, налагодження, система управління версіями та рефакторинг коду. Усі ці функції виходять з коробки. Єдиним недоліком PyCharm є те, що дана програма є доволі ресурсомістком. Потребує не менше 4Гб RAM для комфортної роботи.

Виходячі із вище написаного, було вирішено зупинитися на Pycharm Community. Вона задовільняє раніше поставленим вимогам. Також вона значно пришвидшує розробку програмного забезпечення через дуже потужу систему автодоповнення й має підтримку великої кількості плагінів, які

можуть знадобитися у майбутньому для аналізу моделі планування на основі штучного інтелекту (побудова UML-діаграм класів, робота із файлами різного формату тощо).

2.3.3 Вибір засобів відображення

Оскільки раніше було поставлено задачу незалежності програмного забезпечення від платфоми, то зручно було б представити результати у вигляді простої Web-сторінки. Це дозволить аналізувати результати на будь-якій операційні системі (починаючи від мобільних пристроїв і закінчуючи настільними ПК). Для даної мети у інструментарії мови Python є велика кількість Web-framework`ів [26]. Розглянемо деякі із ним нижче:

1) Django

Це повноцінний інструмент для веб-розробок який є вільним та відкритим. Він досить швидкий. Розробники мають можливість працювати над складними кодами та програмами швидко та гнучко. У Django є багато дивовижних особливостей, що допомогли йому виділитися в порівнянні з іншими рамками Python, наявними на ринку. Деякі його особливості - це адміністрування вмісту, механізм аутентифікації, маршрутизація URL-адрес, механізм роботи з шаблонами та міграція схеми бази даних.

2) Bottle

Це мікро-framework, який відомий своїми мінімалістичними особливостями. Основне його використання - це створення веб-API. Він досить невеликих розмірів; однак його розмір не впливає на його ефективність. Це дійсно одна з найвищих платформ, якій віддають перевагу люди. Цей framework простий і одночасно досить швидкий. «Пляшка» - це мінімалістичний та простий фреймворк, який можна використовувати для створення додатків для особистого використання.

3) Flask

Це дуже невимогливий до ресурсів мікро-framework для розробки web-додатків. Він розроблений таким чином, щоб швидко почати із ним роботу для легкого масштабування складних задач. Надає можливість мінімалістично використовути пам’ять завдяки тому, що він оптимізований і не тягне за собою багато залежностей. Один із найпопулярніших інструментів серед Python розробників.

Із вище переглянутих інструментів для реалізації поставленої задачі підходить будь-який. Слід зупинитися на Flask, оскільки він «найлегший» із засобів та дозволить у короткі строки спроектувати потрібний макет та зекономити на розмірі проекту.

Висновок до розділу 2

В результаті розглянутої інформації у даному розділі щодо вибору й обгронтування засобів реалізації можна прийти до наступних висновків:

1. Вибрано математичну модель планування, виокремлено основні параметри й формули, які вплинуть на подальшу реалізацію поставленого завдання (див. пункт 2.1).
2. Обгрунтовано вибір алгоритма генетичного алгоритму для імплементації його у майбутньому програмному продукті. Описано основні терміни та їх кроки реалізації у відповідності до тематики планування (див. пункт 2.2).
3. Представлено конкретні вимоги щодо засобів й середовища реалізації програмного продукту й проведено аналіз підходящих аналогів й вибрано найкращий (див. пункт 2.3).

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

Базуючись на дослідженому матеріалі й вибраних засобах реалізації можна перейти до етапу розробки програмного продукту. З метою створення гнучкої та настроюваної дослідно-експериментальної моделі планування на основі штучного інтелекту необхідно описати власне основні частини розробленого продукту, функціональні блоки програми та схеми їх взаємодії. Потрібно описати реалізацію моделі навчання, генерації планування, описати перелік характеристик програмних об’єктів.

Предметом розгляду даного розділу є реалізація схеми зображеної на рисунку 3.1.

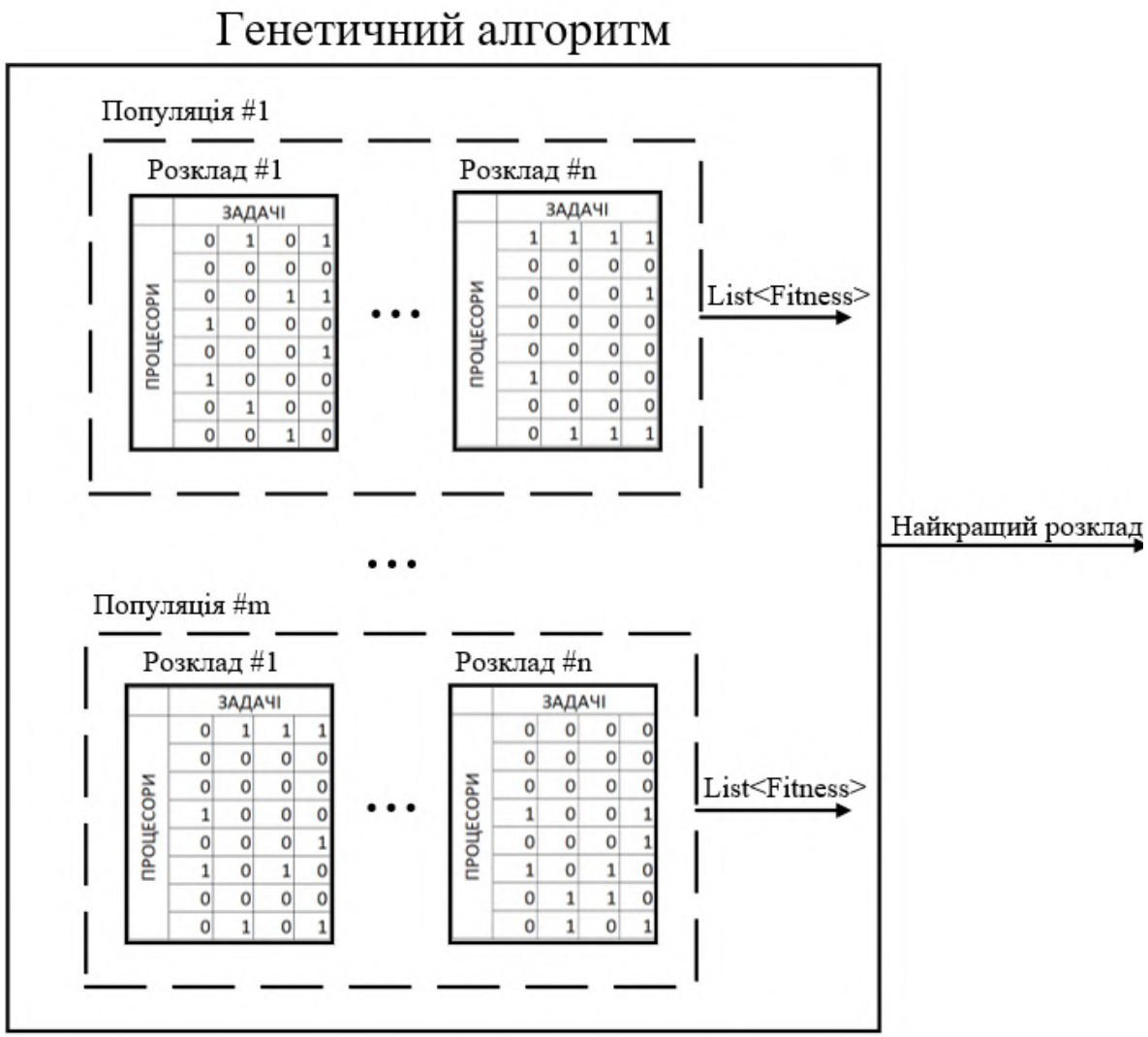


Рис. 3.1 Схема роботи GA

3.1 Розробка програмних компонентів

3.1.1 Клас Task

Клас *Task* (рис. 3.1) представляє собою власне задачу, яка буде плануватися. Більш детально «поняття» задачі було описане раніше, у розділі 1 (пункт 1.1.2). Нижче продемонстрований опис атрибутів (табл. 3.1) й методів класу Task.

Таблиця 3.1 Атрибути класу Task

Назва атрибута	Тип даних	Опис
duration	Number type	Тривалість обробки завдання.
min_completion_time	Number type	Мінімальний час для завершення обробки.
identifier	Number type	Унікальний індектифікатор об’єкта.
name	String type	Строкове представлення об’єкта.
priority	Number type	Пріорітет завдання.
dependencies	List<Task> type	Список задач від яких залежить дана задача.

Єдиний конструктор класу *Task* приймає описані вище поля для ініціалізації об’єкта. По замовчуванню список залежностей відсутній.
Опис основних публічних методів:

- *get_min_completion(self)* - повертає мінімальний час виконання цього завдання на основі мінімальних термінів виконання його залежностей.
- *is_dependency_of(self, other)* - повертає булеве значення *true*, якщо це завдання є залежністю (прямою чи непрямою) від *other*. Інакше – *false*. Як вхідний параметр *other* приймається об’єкт класу *Task*.

- *__repr__(self)* - один із «магічних» методів Python, який є перевизначеним для строкового представлення об’єкту. Зручно використовувати при тестуванні й відлагодженні програми.

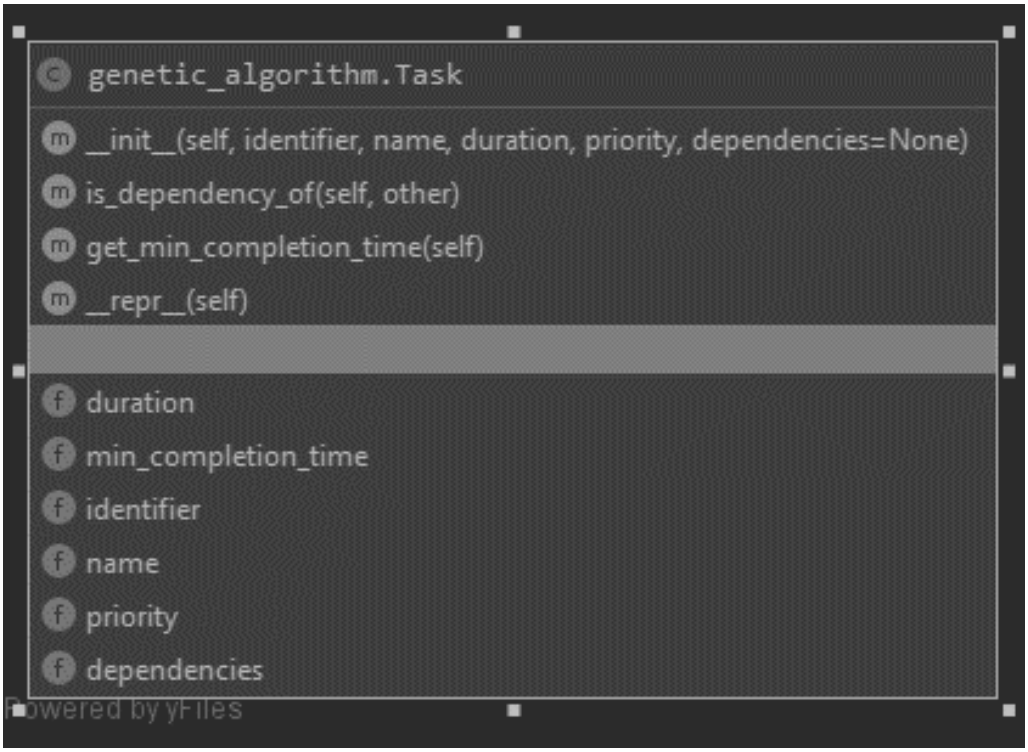


Рис. 3.2 UML діаграма класу *Task*

На рисунку 3.2 можна побачити вище описані методи й атрибути.

3.1.2 Клас *Schedule*

Наступним розробленим компонентом є клас *Schedule*, який інкапсулює у собі логіку планування задач. Необхідний для маніпулюванням об’єктами класу *Task*, використовується як «індивід» популяції. Нижче продемонстрований опис атрибутів (табл. 3.2) й методів класу *Schedule*.

Єдиний конструктор класу Schedule приймає розклад задач на процесори. У більш спрощеному варіанті цей розклад – матриця (див. пункт 2.1) розмірністю n x m, де значеннями виступають об’єкти класа Task. По замовчуванню цей параметр відсутній.

Опис основних публічних методів:

- *min_processor_schedule_length(self)* – повертає найменшу кількість завдань у всередині розкладу серед всіх процесорів.

Таблиця 3.2 Атрибути класу Schedule

Назва атрибута	Тип даних	Опис
task_dependency_set_map	Map<Task, Set<Task>> type	Пара ключ-значення, яка ставить у відповідність задачу як ключ і множину задежностей.
processor_schedules	List<List<Task>> type	Список розкладів на кожному із процесорів. Відповідний індекс списка – індекс процесора.
task_completion_map	Map<Task, Integer> type	Пара ключ-знначення, яка ставить у відповідність завдання як ключ і час його завершення як значення.

- *has_unique_tasks(self)* – повертає булеве значення *true*, якщо в розкладі немає більше одного і того ж завдання. Інакше – *false*.
- *has_direct_dependency_violatation(self)* – повертає булеве значення *true*, якщо в будь-якому процесорі завдання виконується перед однією з його залежностей. Інакше – *false*.
- *get_task_location(self, task)* – повертає індекси процесора та завдання будь-якої заданої задачі. None, якщо завдання відсутнє в цьому графіку. Як вхідний параметер *task* приймається об’єкт класу *Task*.
- *get_dependency_set(self, task)* – повертає набір усіх завдань, після яких дане завдання має бути виконано в цьому графіку. Як вхідний

параметер task приймається об’єкт класу Task.

- *calculate_task_completion(self, processor_index, task_index)* – рекурсивно визначає, коли завдання виконається. Як вхідні парамететри *processor_index, task_index* приймаються цілі числа, індекси для матриці *processor_schedules*.
- *get_task_completion_map(self)* - для кожного завдання визначає, коли воно виконається й формує *task_completion_map*.
- *calculate_time_grid(self, total_time)* - обчислює й формує таблицю часових інтервалів під час яких виконуються завдання. Як вхідний параметер *total_time* приймається ціле число, яка буде визначати межі сформованої таблиці.
- *clone(self)* – робить копію даного об’єкт класу *Schedule*. Є важливим методом для реалізації копій, що будуть використовуватися у генетичному алгоритмі.
- *reproduce(self, other)* – враховуючи переданий параметр *other* схрещує їх і повертає список із двох нових об’єктів класу *Schedule*. Як вхідний параметер *other* приймається об’єктів класу *Schedule* із яким буде схрещуватися даних об’єкт.
- *mutate(self)* – вибирає випадковим чином 1 задачу й вставляє її у випадковий індекс у графіку.

3.1.3 Клас *GeneticTaskScheduler*

Наступним розробленим компонентом є клас *GeneticTaskScheduler*, який реалізовує у собі власне логіку самого генетичного алгоритму. Алгоритми й підходи використані у цьому класі були розглянуті у розділі 2. Загальна схема роботи класу була описана у вступній частині розділу 3 на рис. 3.1. Нижче продемонстрований опис атрибутів (табл. 3.3) й методів класу *GeneticTaskScheduler*.

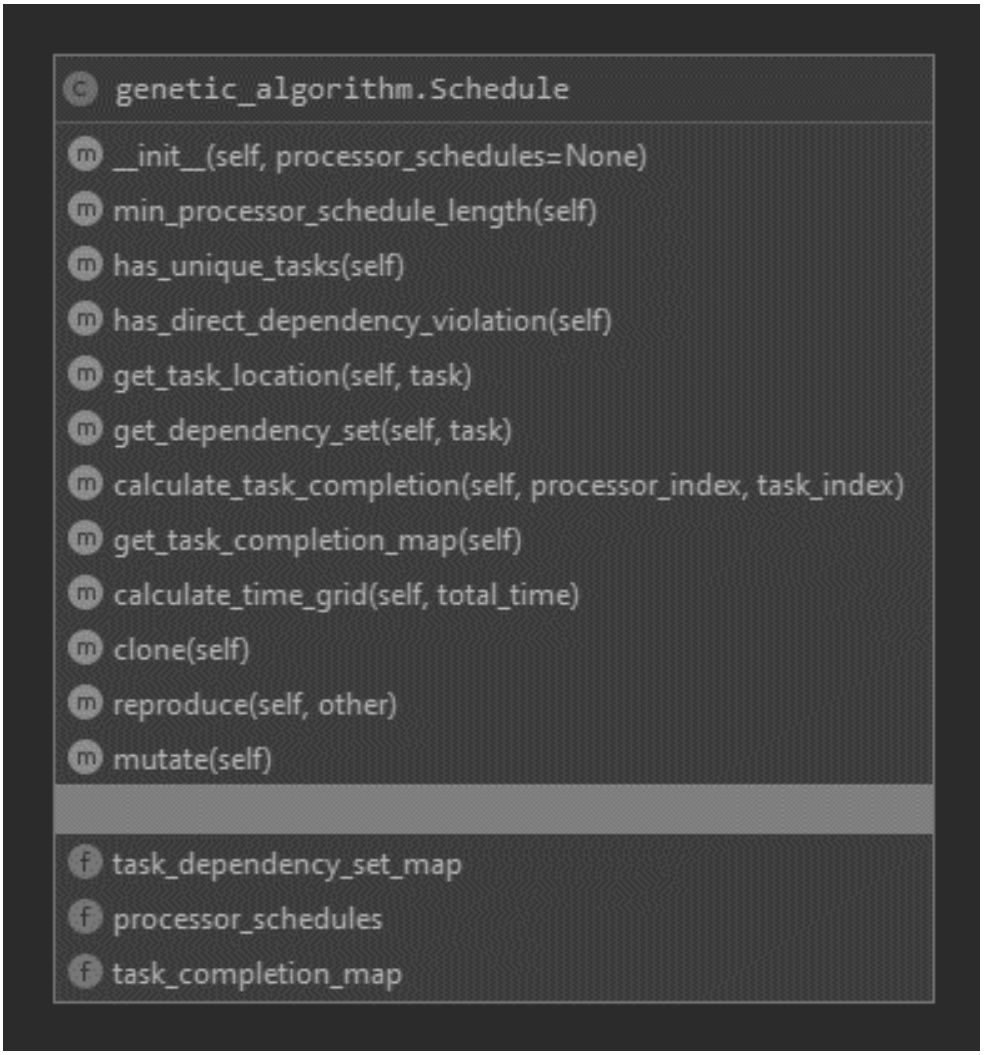


Рис. 3.3 UML діаграма класу *Schedule*

Єдиний конструктор класу *GeneticTaskScheduler* приймає список задач. Опис основних публічних методів:

- *initialize(self, num_processors, population_size, total_time)* – цей метод, враховуючи список завдань, кількість процесорів та чисельність популяції, створить початкову популяції. Як вхідні параметри примаються цілі невід'ємні числа.
- *_get_task(self, identifier)* – цей метод видає завдання за його ідентифікатором. Вхідний параметр *identifier* є цілим числом.
- *select(self, old_population) – це метод селекції. Випадковим чином вибирає індивідів, які виживають, для відтворення залежно від функції fitness.*

Таблиця 3.3 Атрибути класу GeneticTaskScheduler

Назва атрибута	Тип даних	Опис
total_time_bound	Integer type	Верхня межа виконання.
priority_flowtime_bound	Integer type	Верхня межа виконання із урахуванням пріорітетів.
total_time	Integer type	Загальний час виконання.
tasks	List<Task> type	Список переданих задач.

- *reproduce(self, population)* – це метод схрещування. Зважаючи на величину популяції, буде випадковим чином відбирати індивіди для відтворення та додавати нових потомків до популяції. Вхідним параметром *population* є список об’єктів класу *Schedule*.
- *mutate(self, population)* – цей метод, аналізуючи популяцію, буде випадково відбирати індивідів для мутації. Вхідним параметром *population* є список об’єктів класу Schedule.
- *fitness(self, population)* – цей метод обчислює перелік значень fitness-функції для популяції. Вхідним параметром population є список об’єктів класу Schedule.
- *schedule_tasks(self, num_processors, generations, total_time)* – за заданим списком обмежень ця функція починає роботу генетичного алгоритму, щодо пошуку найкращого рішення розкладу для заданого переліку задач.

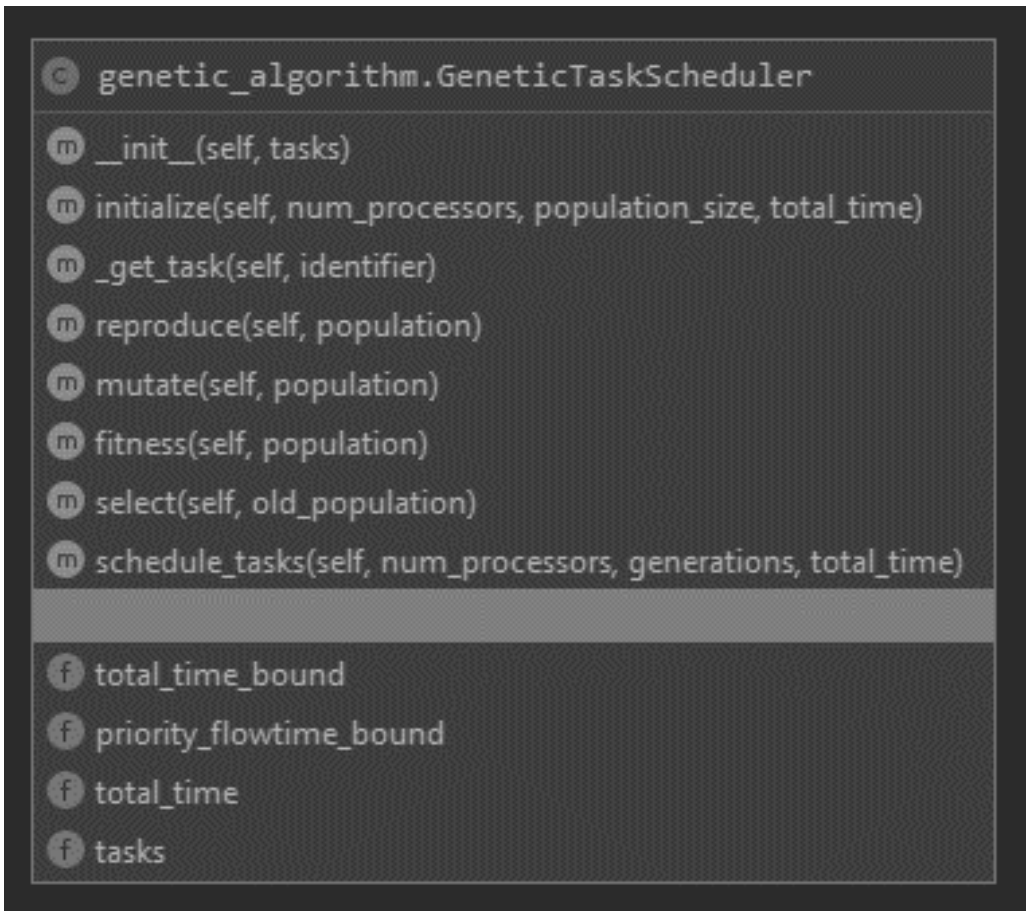


Рис. 3.4 UML діаграма класу *GeneticTaskScheduler*

На рисунку 3.4 можна побачити вище описані методи й атрибути.

3.1.4 Точка входу для віддаленого доступу

Для того, щоб мати можливість використовувати розроблене ПЗ віддалено необхідно прописати API за допомогою якого можна буде отримати власне доступ до даних. У Flask є спеціальні декоратори для вирішення цієї задачі. Для цього за допомогою декоратора *@app.route()* вкажуємо необхідний шлях і метод, який буде займатися обробкою запиту.

Маршрутизація на головну сторінку :

```
@app.route('/')
def main():
    return render_template('main.html')
```

Таким чином при зверненні на адрес *<application_domain>/* користувач буде потрапляти на сторінку *main.html*.

Для того, щоб відобразити резульати необхідно обробити дані, які вводитиме користувач. Переважно це робиться за домогою обробки форми й відправки HTTP метода *POST*.

Маршрутизація на результати планування:

```
@app.route('/schedule', methods=['POST'])
def schedule():
    # Обробка запиту клієнта
    data = request.json
    ...
    # Алгоритм планування
    gen_alg = genetic_algorithm.GeneticTaskScheduler(tasks)
    schedule = gen_alg.schedule_tasks(processors, generations, total_time)
    # Форматування результату
    ...
    # Відправлення результату клієнту
    return json.dumps(schedule)
```

3.2 Інструкція встановлення програми на локальній машині

Задля успішного встановлення й тестування розробленого ПЗ необхідно, щоб виконувалися вимоги визначені у технічному завданні. Якщо вимоги виконані, то переходимо до наступних кроків.

У директорії проекту, де буде розміщений проект клонуємо проект по посиланню [27] (рис. 3.5).

Встановлюємо необхідні залежності (див. рис. 3.6) із консолі командою:

```
>pip install requirements.txt
```

```
C:\Users\sanve\PycharmProjects\Sample>git clone https://github.com/Sanverik/SchedulerAI
Cloning into 'SchedulerAI'...
remote: Enumerating objects: 35, done.
remote: Counting objects: 100% (35/35), done.
remote: Compressing objects: 100% (28/28), done.
remote: Total 35 (delta 4), reused 30 (delta 2), pack-reused 0
Unpacking objects: 100% (35/35), done.
```

Рис. 3.5 Логи клонованого проекту

```
(venv) C:\Users\sanve\PycharmProjects\Sa>pip install -r requirements.txt
Collecting Flask==1.0.2 (from -r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/7f/e7/08578774ed4536d3242b14dacb4696386634607af824ea997202cd0edb4b/Flask-1.0.2-py2.py3-none-any.whl
Collecting gunicorn==19.9.0 (from -r requirements.txt (line 2))
  Using cached https://files.pythonhosted.org/packages/8c/da/b8dd8deb741bff556db53902d4706774c8e1e67265f69528c14c003644e6/gunicorn-19.9.0-py2.py3-none-any.whl
Collecting Jinja2>=2.10 (from Flask==1.0.2->-r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/30/9e/f663a2aa66a09d838042ae1a2c5659828bb9b41ea3a6efa20a20fd92b121/Jinja2-2.11.2-py2.py3-none-any.whl
Collecting Werkzeug>=0.14 (from Flask==1.0.2->-r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/cc/94/5f7079a0e00bd6863ef8f1da638721e9da21e5bacee597595b318f71d62e/Werkzeug-1.0.1-py2.py3-none-any.whl
Collecting click>=5.1 (from Flask==1.0.2->-r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/d2/3d/fa76db83bf75c4f8d338c2fd15c8d33fdd7ad23a9b5e57eb6c5de26b430e/click-7.1.2-py2.py3-none-any.whl
Collecting itsdangerous>=0.24 (from Flask==1.0.2->-r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/76/ae/44b03b253d6fade317f32c24d100b3b35c2239807046a4c953c7b89fa49e/itsdangerous-1.1.0-py2.py3-none-any.whl
Collecting MarkupSafe>=0.23 (from Jinja2>=2.10->Flask==1.0.2->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/b9/2e/64db92e53b86efccfaea71321f597fa2e1b2bd3853d8ce658568f7a13094/MarkupSafe-1.1.1.tar.gz
Installing collected packages: MarkupSafe, Jinja2, Werkzeug, click, itsdangerous, Flask, gunicorn
  Running setup.py install for MarkupSafe ... done
Successfully installed Flask-1.0.2 Jinja2-2.11.2 MarkupSafe-1.1.1 Werkzeug-1.0.1 click-7.1.2 gunicorn-19.9.0 itsdangerous-1.1.0
```

Рис. 3.6 Логи встановлених залежностей

Запускаємо програму (рис. 3.7) через наступну команду із консолі:

>python app.py

```
C:\Users\sanve\OneDrive\Desktop\SchedulerAI>python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 129-168-912
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Рис. 3.7 – Логи запуску програми

Тепер можна перейти за посиланням http://127.0.0.1:5000/ й використовувати програму.

Висновок до розділу 3

В результаті проведеної роботи щодо розробки програмного забезпечення моделі планування на основі штучного інтелекту можна виокремити наступне:

- Реалізовано раніше обгрунтовану модель генетичного алгоритму (див. розділ 2, пункт 2.1).
- Наведено UML діаграми розроблених класів та детально описано їх атрибути та методи (див. пункт 3.1).
- Реалізовано модель віддаленого доступу до результатів планування за допомогою Flask (див. пункт 3.1.4).
- Описано детальну інструкцію користувача щодо встановлення розробленого ПЗ (див. пункт 3.2).

РОЗДІЛ 4. РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ

Базуючись на розробленій програмній моделі планування метою даного розділу є показати результати її роботи. Також потрібно провести порівняльну характеристику розробленого алгоритму планування на основі штучного інтелекту. Описати переваги й недоліки розробленого методу на прикладах.

4.1 Програмний інтерфейс

На рисунку 4.1 зображений розроблений програмний інтерфейс користувача. Це головна сторінка *main.html*. У полях вводу можна побачити дані за замовчуванням:

- 10 генерацій генетичного алгоритму;
- 3 процесори;
- Обмеженння по часу планування – 10 одиниць.

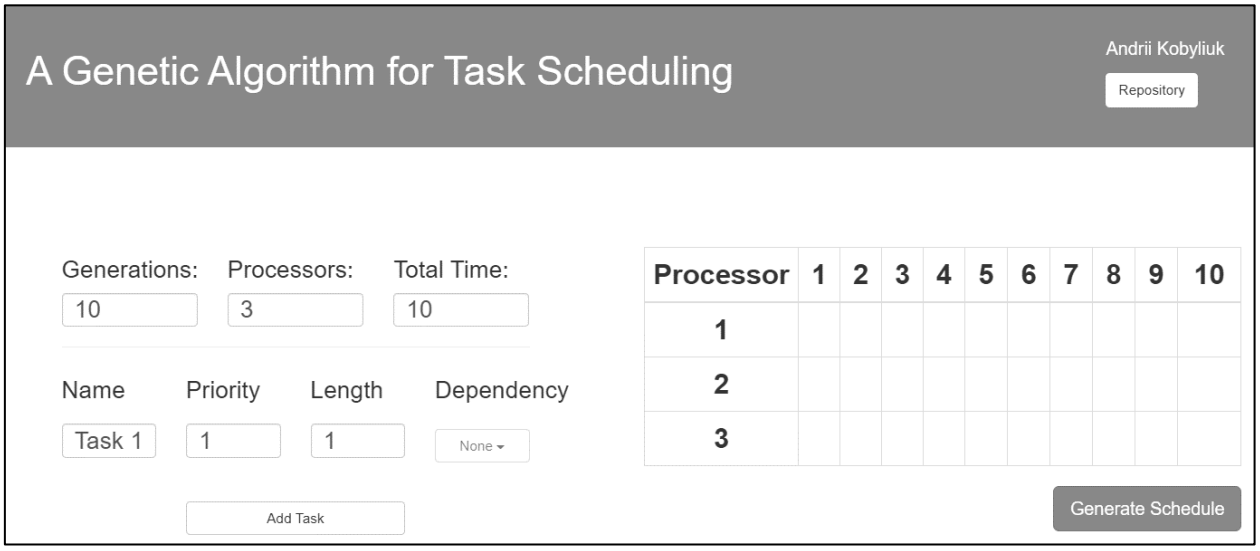


Рис. 4.1 – Програмний інтерфейс

Натиснувши на відповідну кнопку *Repository* можна перейти за посиланням [27] на сторінку із програмним кодом (див. рис. 4.2), а також перейти на сторінку автора на https://github.com.

Розглянемо можливість користувача оперувати вхідними даними програми. Для цього передбачена наступна форма (див. рис. 4.3).

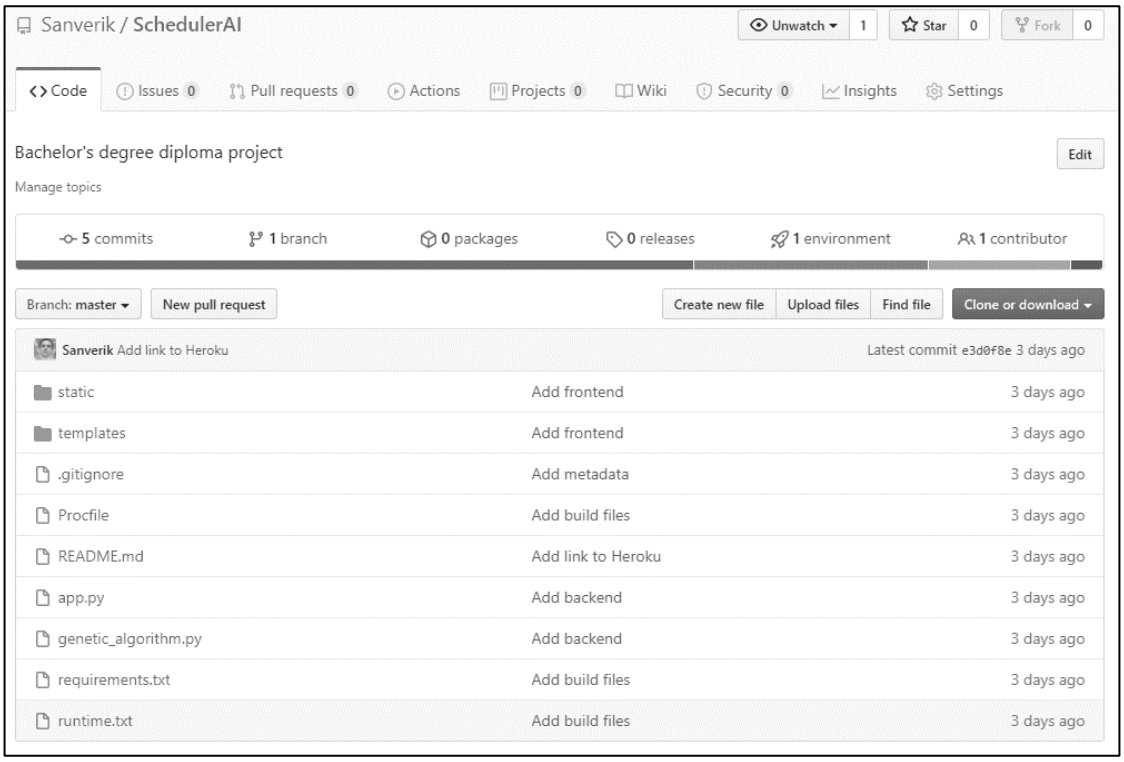


Рис. 4.2 – Дерево проекту на GitHub

Поля, які безпосередньо задають характеристики планування:

- Поле *Generations* – вхідні дані щодо кількості генерацій генетичного алгоритму.
- Поле *Processors* – вхідні дані щодо кількості процесорів для обробки задач.
- Поле *Total Time* – вхідні дані щодо крайнього часу завершення графіку планування.

Поля, які безпосередньо задають характеристики кожної окремої задачі:

- Поле *Name* – мітка, яка буде ідектифікувати завдання.
- Поле *Priority* – числове значення пріорітету відвовідного завдання (менше значення – більший пріорітет).
- Поле *Length* – задає кількість одиниць часу для обробки завдання.
- Поле *Dependency* – задає залежність задачі від попередніх. По замовчуваню залежність відсутня.

Кнопка *Add Task* відповідно додає нове завдання у чергу для можливості подальшого редагування.

Далі розглянемо можливість користувача отримати й бачити результат отриманого планування по вхідним даним, які були введені раніше. Для цього передбачена наступна елемент відображення (див. рис. 4.4).

Для генерації графіку планування необхідно просто натиснути кнопку *Generate schedule* й очікувати результат. У залежності від кількості заданих ітерацій генетичного алгоритму та складності взаємодії задач потрібно очікувати 1-2 секунди на проведення необхідних обчислень й відповідь програми.

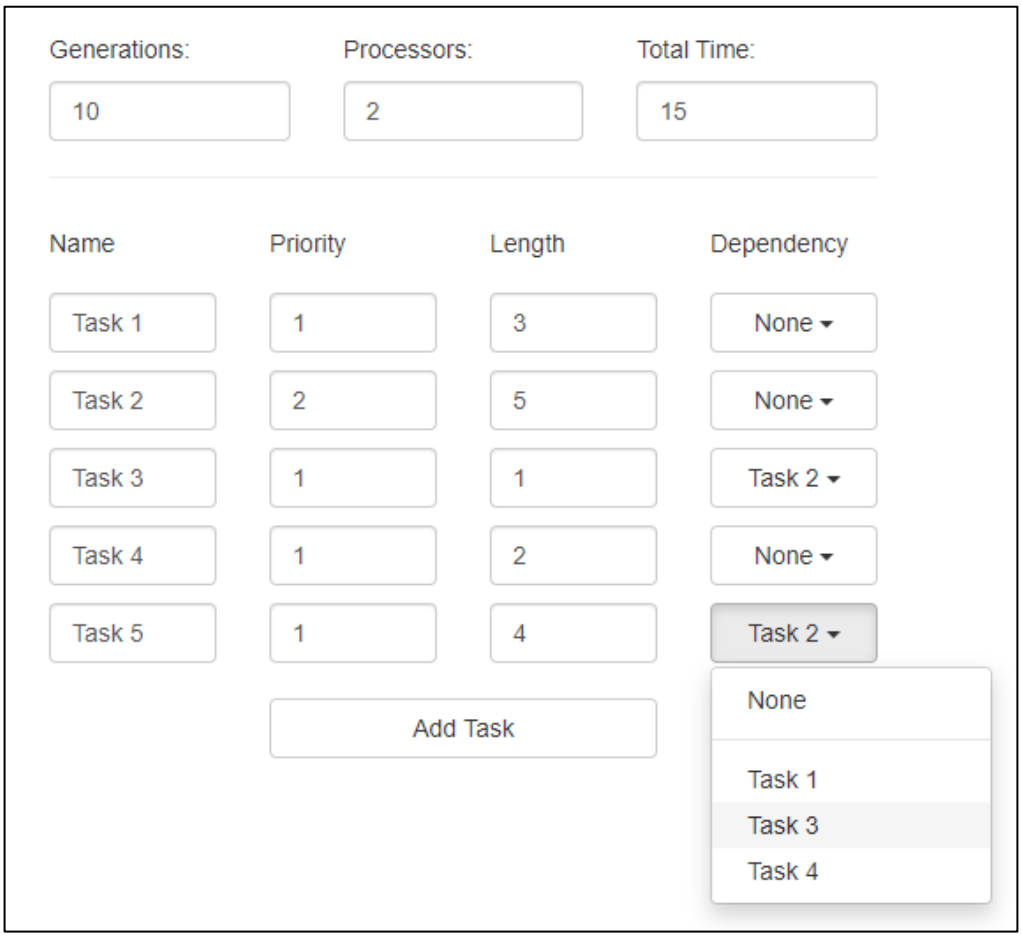


Рис. 4.3 – Форма вхідних даних

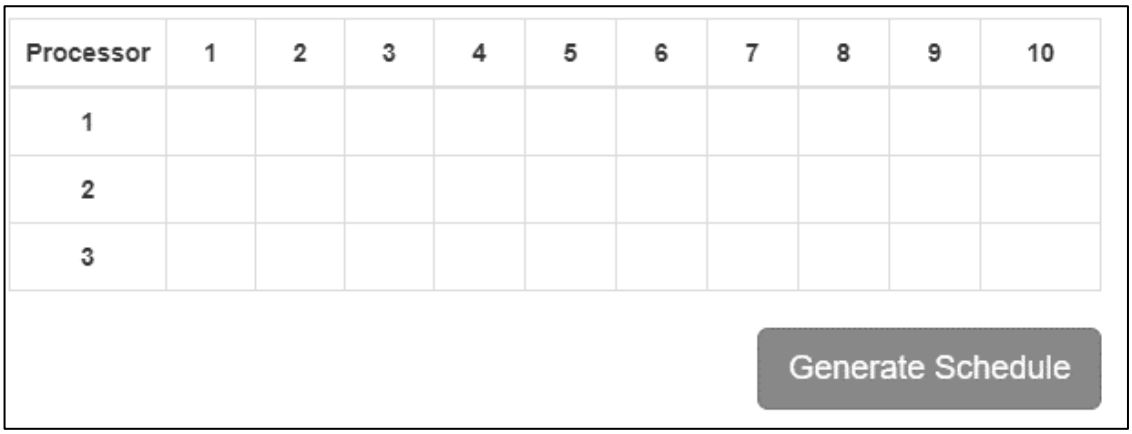


Рис. 4.4 – Форма вихідних даних

У разі винекнення непередбачених помилок під час роботи програми передбачено сповіщення користувача щодо виникнення помилки. Серед таких помилок може бути некоректні вхідні дані, помилки під час роботи генетичного алгоритму тощо. Приклад сповіщення про помилку можна побачити на рисунку 4.5.

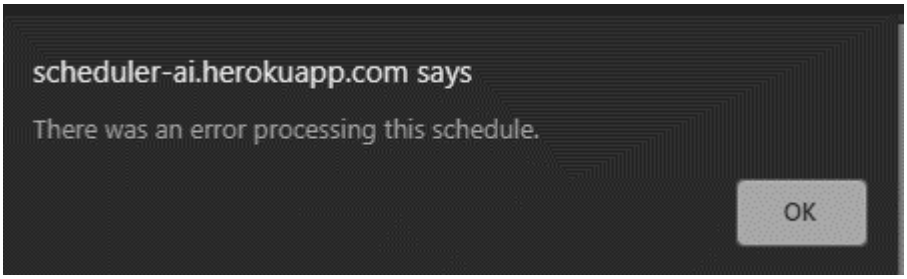


Рис. 4.5 – Обробка помилки

У разі успішного виконання планування користувач побачить результат подібний до рисунку 4.6. Вхідними даними для цього моделювання є дані із рисунку 4.3.

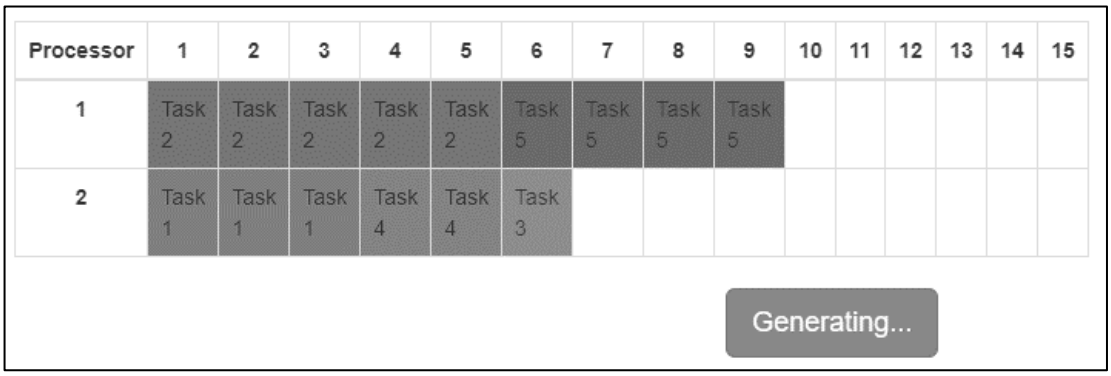


Рис. 4.6 – Форма вихідних даних

Слід зазначити, що через специфіку генетичного алгоритму можна отримати різні графіки натиснувши кнопку *Generate schedule* декілька раз. Це перш за все пов’язано сам алгоритм у більшій мірі базується на випадкових величинах. Цікаво, що при зменшення кількості генерацій до менш, ніж п’яти (залежисть від кількості й складності взаємозв’язків між задачами) часто можна отримати неоптимальний графік. Як приклад було зменшено кількість генерацій генетичного алгоритму до 2. Результати зображені на рисунку 4.7. Видно, що у порівнянні із попереднім дослідом кінцевий час завершення погіршився на 2 одиниці.

Рис. 4.7 – Неоптимальне рішення

4.2 Тестування програми

Для тестування програми було розроблено три різні моделі топології залежності задач. Метою проведення даних тестувань є демонстрування роботи розробленого планування й, можливо, виявлення недоліків розробленого підходу, шляхом оцінки якості й часу виконання програми.

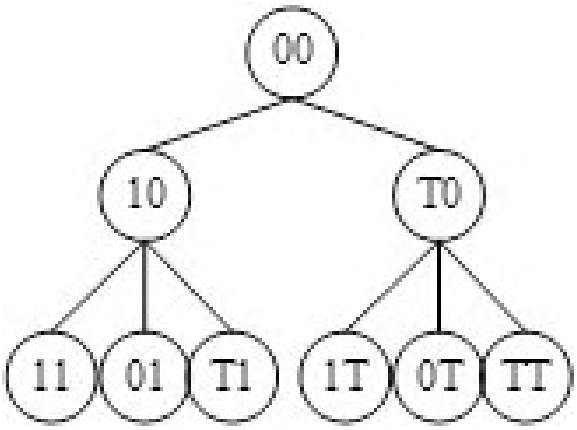


Рис. 4.8 – Модель №1

Для даного графа (див. рис. 4.8) експерементальним шляхом було виявлено, що достатьо 5 генерації генетичного алгоритму задля досягнення повтавленої цілі. Із таблиці 4.1 видно, що збільшення кількості процесорів веде до скорочення загального часу роботи алгоритму планування. У кожному тесті було досягнуто оптимальне рішення.

Таблиця 4.1 Результати тестування моделі №1

№ тесту	Кількість процесорів	Час очікування	Момент виконання	Коефіцієнт завантаженості
1.1	1	0.042 с	9	1.00
1.2	2	0.020 с	5	0.90
1.3	4	0.015 с	4	0.56
1.4	6	0.015 с	3	0.50

Processor	1	2	3	4
1			1T	
2		10	0T	
3	00	T0	TT	
4			11	
5			01	
6			T1	

Рис. 4.9 – Оптимальний результат для моделі №1

У наступному тесті граф взаємозв’язків задач було дещо ускладнено (див. рис. 4. 10).

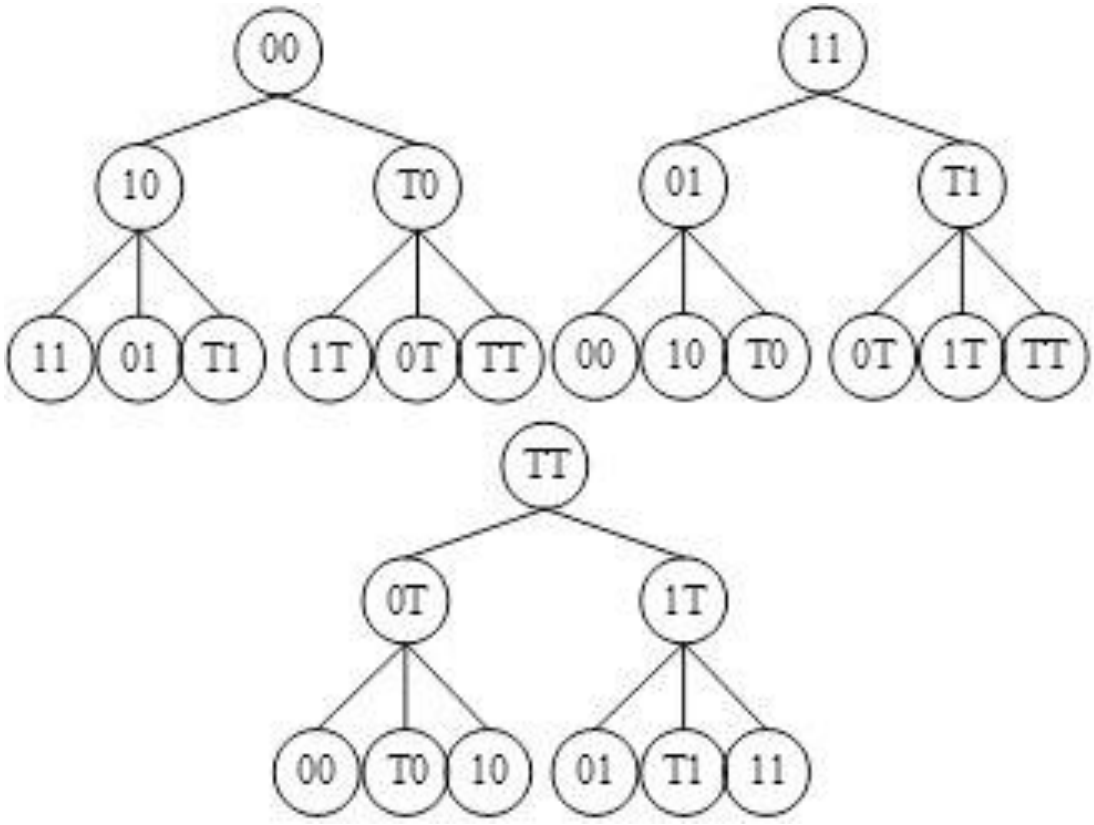


Рис. 4.10 – Модель №2

У досліді №3 генетичний алгоритм однозначно не може визачити оптиальне планування (див. рис. 4.11). Помилка може бути у діапазоні 1-2 тактів по завершенню.

Processor	1	2	3	4	5	6	7	8	9	10
1	00		01	TT	TT		T0	11		
2		10	T1	01	T0	0T	10			
3		T0	1T	T1	0T	1T	01			
4			11	0T	00	10	1T	00	T1	

Processor	1	2	3	4	5	6	7	8	9	10
1	00		01	TT	TT		T0	11		
2		10	T1	01	T0	0T	10			
3		T0	1T	T1	10	0T	1T	01		
4			11	0T	00	1T	00	T1		

Рис. 4.11 – Приклад неоптимального планування програми

Як видно із таблиці №4.2, час очікування на виконання планування за розроблемим алгоритмом зріс у декілька раз порівнюючи із попереднім дослідом на графі із моделі №1.

Таблиця 4.2 Результати тестування моделі №2

№ тесту	Кількість процесорів	Час очікування	Момент виконання	Коефіцієнт завантаженості
2.1	1	7.346 с	25	1.00
2.2	2	5.867 с	13	0.96
2.3	4	3.434 с	8	0.69
2.4	6	3.102 с	7	0.60

Ну і наостанок протестуємо модель, що зображена на рисунку 4.12, щоб остаточно переконатися у результатах тестування й зробити відповідні висновки. Як можна замітити вона є розширеною версією попередньої моделі тестування.

Таблиця 4.3 Результати тестування моделі №3

№ тесту	Кількість процесорів	Час очікування	Момент виконання	Коефіцієнт завантаженості
3.1	1	>2 хв	79	1.00
3.2	2	>2 хв	39	0.98
3.3	4	>2 хв	22	0.75
3.4	6	>2 хв	20	0.65

У даному наборі тестувань все більш часто виникають проблеми із пошуком оптимального рішення. Для покращення результатів є можливість збільшення кількості генерації алгоритму, проте результати не можна вважати детерміністичними. Час виконання є занадто великим (див. табл. 4.3), щоб говорити про практичне застосування розробленого алгоритму на даній моделі й даній кількості задач.

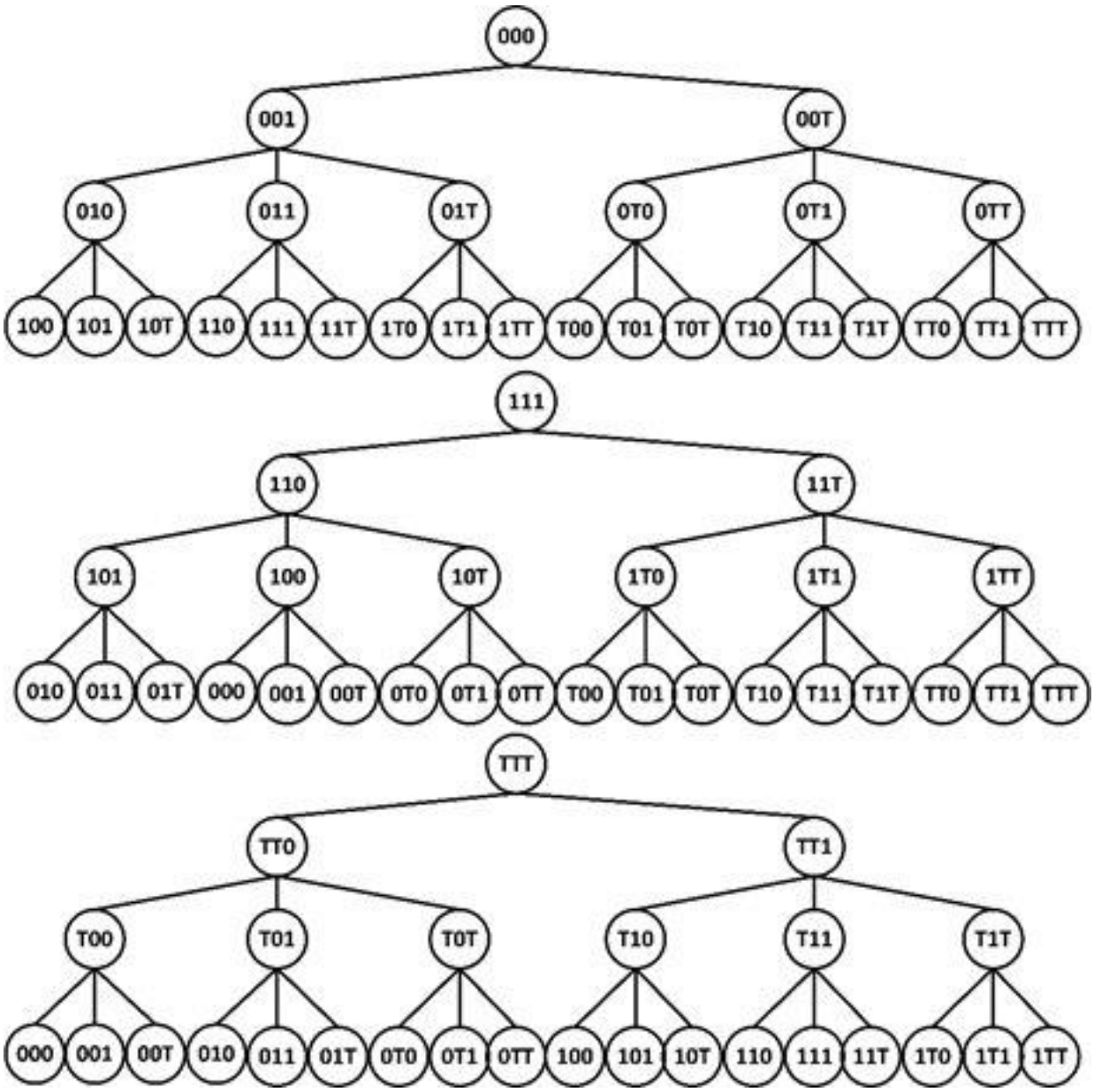


Рис. 4.10 – Модель №3

Processor	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	000		011	110	1T1	T0T	TT0	11T	1T0	011	00T	T00	T11	TTT		T01	T1T	010	0T1	10T	1T0		
2		001	01T	100	111	1TT	T10	TT1	101	1T1	01T	0T0	T01	T1T	TT0	T0T		000	011	0TT	110	1T1	
3		00T	0T0	101	11T	T00	T11	TTT	100	1TT	000	0T1	T0T	TT0	TT1	T10		001	01T	100	111	1TT	
4			010	0T1	10T	0TT	1T0	T01	T1T	110	10T	010	001	0TT	T10	TT1	T00	T11	00T	0T0	101	11T	

Generate Schedule

Рис. 4.11 – Приклад планування для моделі №3

4.3 Ефективність розробленого підходу

Порівняти ефективність розробленого алгоритму можна порівнюючи із недавно зробленою пов’язаною роботою [28]. У цій роботі розглядається подібна подібна тема планування, але за домогою інших методик наявних інструментів штучного інтелекту. Зокрема для реалізації планування використовується дворівнева згорткова нейронна мережа, яка вибирає найкращий евристичний алгоритм (див. рис. 4.12).

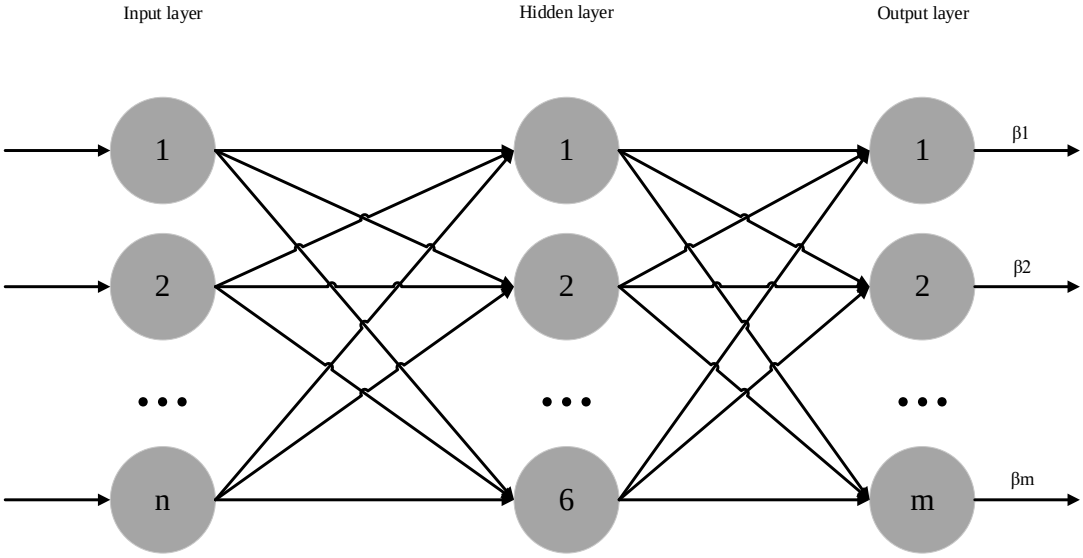


Рис. 4.12 – Один із рівнів нейромережі

Проаналізовано структуру й параметри двох нейромереж, які показали показали прийнятні результати, щодо вибору найкращого алгоритму для планування. Приблизний відсоток правильності був вище 85%. Рішення створити комбінацію із двох мереж призвело до помітних покращень у точності прогнозування. Недоліком розробленого підходу було те, що для кожного окремого графу взаємодії задач нейромережу необхідно було реконфігорувати й перенавчати.

Таблиця 4.4 Результати тестування на нейромережі

№ тесту	Кількість процесорів	Час очікування	Момент виконання	Коефіцієнт завантаженості
4.1	1	~6 с	79	1.00
4.2	2	~8 с	39	0.98
4.3	4	~12 с	22	0.75
4.4	6	~14 с	20	0.65

У випадку розробленого генетичного алгоритму цей недолік усувається. Є можливість скласти графік планування для будь-якого графу задач без зміни коду програми. Також є незначний приріст у плані «навчання» алгоритму у порівнянні із вище зазначеним підходом.

Звичайно, що даний алгоритм не може вважатися універсальним. У нього є свої обмеження на застосування. Наприклад, у системах реального часу, де є вимога на часткові рішення у найкоротші проміжки часу він може знайти своє застосування тощо.

Для невеликих наборів задач, можливо, навіть більш ефективно буде використовувати точні методи планування, то за детермінований час можна знайти оптимальне рішення.

Коли відома наперед структура графа задач, або є більше інформафії про планування, то можна вибрати окрему евристику, яка буде справлятися із задачею ефективніше за розроблений підхід.

Висновок до розділу 4

В результаті проведеної роботи щодо моделювання розробленого програмного забезпечення моделі планування на основі штучного інтелекту можна виокремити наступне:

- Покроково описано програмний інтерфейс розробленої програми (див. пункт 4.1).
- Наведено опис функціоналу та продемонстровано результат роботи програми (див. пункт 4.1).
- Протестовано розроблений алгоритм на різних вхідних моделях задач (див. пункт 4.2).
- Порівняно реалізований алгоритм планування на основі штучного інтелекту порівнюючи його із наявними методами реалізації поставленої задачі (див. пункт 4.3).

ВИСНОВОК

У першому розділі даної роботи було розглянуно предметну область й опрацьовано теоретичну базу, що стосувалася даної теми роботи. Було продемонстровано підходи, щодо планування задач із обмеженими ресурсами та методи вирішення поставленої проблеми. Зокрема, розглянуто підходи, що стосуються штучного інтелекту (метод «імітації підпалу» та сімейство генетичних алгоритмів). Було окреслено область подальної роботи у рамках вибраної теми.

У другому розділі даної роботи було розглянуто методи й інструменти для реалізації раніше поставленої задачі. Зокрема описано покроково реалізацію генетичного алгоритму, описано головні поняття й термінани, якими оперують при роботі із даним методом. Вибрано базовий інструментарій щодо подальшої розробки програмного забезпечення моделі планування на основі штучного інтелекту. Обгрунтовано, чому саме такі технології були вибрані.

У третьому розділі даної роботи було покроково продемонстровано етапи розробки програмного забезпечення. Наведено опис й перелік програмних компонентів. Для успішного тестування ПЗ було наведено інструкцію користувача по роботі із програмним кодом.

У четвертому розділі продемонстровано власне роботу розробленої моделі планування на основі штучного інтелекту. Проведено порівняльну характеристику розробленого алгоритму на різних типах моделей взаємодії задач.

Результати, представлені в даній роботі, показують успішне застосування штучного інтелекту, а саме генетичних алгоритмів, до проблеми планування. Більше того, функціонування пріоритетних напрямків, створених GP, показує багатообіцяючі результати в статичному та динамічному середовищі, де у багатьох інших евристик та підходів не вистачає при

вирішенні можливостей та продуктивності. У порівнянні з існуючою евристикою, у більшості випадків досягнуті результати генерують значно кращі графіки, представляючи конкурентний підхід до вирішення задачі планування.

Негативною стороною генетичних алгоритмів виявилося те, що часто вони знаходять тільки локальне рішення, що може призвести до неоптимальних результатів. Також обмеженням генетичного алгоритму є те, що для кожної предметрої області необхідно описувати й реалізовувати нову модель.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Aarts E. Simulated annealing and boltzmann machines / E. Aarts, J. Korst., 1988.
2. Boctor F. F. "Heuristicts for scheduling with resource restrictions and several resource-duration modes" / Boctor. // International Journal of Production Research. – 1993. – №31. – С. 2547.
3. Simulated annealing [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Simulated_annealing.
4. Davis L. Job Shop Scheduling with Genetic Algorithms / Lawrence Davis. // Proceedings of the 1st International Conference on Genetic Algorithms. – 1985. – С. 136–140.
5. Bruns R. Knowledge-based scheduling systems in industry and medicine / R. Bruns, J. Sauer. // IEEE Expert. – 1997. – С. 24 – 31.
6. Bagchi T. P. Multiobjective Scheduling by Genetic Algorithms / Tapan Bagchi. – New York: Springer Science, 1998. – 351 с.
7. Hasband P. The Natural Way to Evolve Hardware / P. Hasband, I. Harvey, T. Adrian. – 1996.
8. Powerful Examples Of Artificial Intelligence In Use Today [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://www.forbes.com/sites/robertadams/2017/01/10/10-powerful-examples-of-artificial-intelligence-in-use-today/#17011af1420d.
9. Pinedo M. L. Scheduling : Theory, Algorithms, and Systems / Michael L. Pinedo. – New York: Springer Science, 2008. – 662 с. – (Prentice Hall). – (978-0-387-78935-4).
10. Ullman J. D. Np-complete Scheduling Problems / J. D. Ullman. // Journal of Computer and System Sciences. – 1973. – №10. – С. 384—393.
11. Grosche T. Computational Intelligence in Integrated Airline Scheduling / Tobias Grosche., 2009. – (Springer). – (978-3-540-89886-3).

12. Randolph H. E. Handbook of Healthcare System Scheduling / Hall E. Randolph., 2012. – (78-1-4614-1734-7).
13. Briskorn D. Sports Leagues Scheduling / Dirk Briskorn., 2008. – (Prentice Hall).
14. Leung J. Handbook of Scheduling: Algorithms, Models, and Performance Analysis / J. Leung, J. Anderson. – New York, 2004. – 1157 с.
15. Project scheduling with limited resources using a genetic algorithm. // International Journal of Project Management. – 2010. – С. 619–628
16. Job Shop Scheduling Problem (JSSP): An Overview [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://medium.com/datadriveninvestor/job-shop-scheduling-problem-jssp-an-overview-cd99970a02f8.
17. James E Kelley Jr and Morgan R Walker. Critical-path planning and scheduling. In Papers presented at the December 1-3, 1959, eastern joint IRE AIEE-ACM computer conference, pages 160–173. ACM, 1959.
18. Patterson J. H. A comprasion of exact approaches for solving the multiple constrained resource project scheduling problem / Patterson., 1984. – 867 с.
19. Held–Karp algorithm [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Held%E2%80%93Karp_algorithm.
20. Bader D. A. Parallel Algorithm Design for Branch and Bound / D. A. Bader, W. E. Hart, C. A. Phillips., 2004. – (Tutorials on Emerging Methodologies and Applications in Operations Research).
21. Sprecher A. Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems / A. Sprecher, A. Drexl., 1996. – 1703 с. – (Management Science). – (41).
22. Stinson, J.P., Davis, E.W., Khumawala, B.M.: Multiple resource-constrained scheduling using branch-and-bound. AIIE Trans. 10(3), 252–259 (1978)
23. Harvey W. D. Limited Discrepancy Search / W. D. Harvey, M. L. Ginsberg. // University of Oregon Eugene. – 1994.

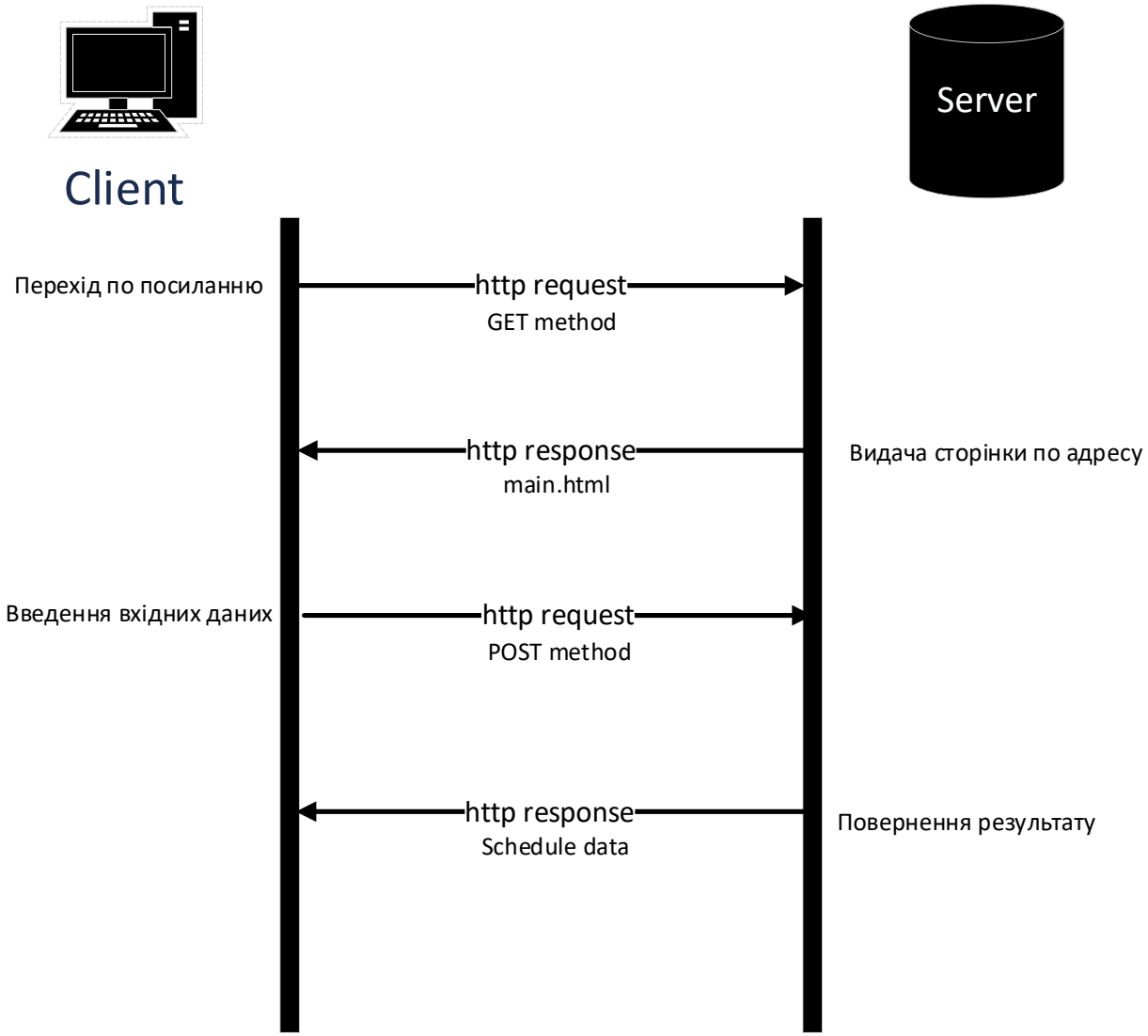
24. 5 Best Programming Languages for Artificial Intelligence in 2020 [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://dev.to/saikrishna_iam/top-5-best-programming-languages-for-artificial-intelligence-in-2020-5ghj.
25. Python IDEs and Code Editors [Електронний ресурс] – Режим доступу до ресурсу: https://www.programiz.com/python-programming/ide.
26. Top 10 Trending Python Web Frameworks In 2020 [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://technostacks.com/blog/python-web-frameworks.
27. SchedulerAI [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/Sanverik/SchedulerAI
28. Artificial Intelligence-based Scheduling. // The International Conference on Security, Fault Tolerance, Intelligence. – 2020. – С. 5.
29. Wall M. W. A Genetic Algorithm for Resource-Constrained Scheduling / Matthew Wall Wall. // Department of Mechanical Engineering. – 1996.
30. Genetic Algorithms [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/genetic-algorithms/.
31. Brucker P. Scheduling Algorithms / Pete Brucker. – Berlin: Springer, 2006. – 378 с. – (978-3-540-69515-8).
32. Basic Structure of Genetic Algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/figure/Basic-Structure-of-Genetic-Algorithm_fig1_261468690.
33. A price-based mechanism for online buyer coalition by genetic algorithms [Електронний ресурс] – Режим доступу до ресурсу: http://www.ijicic.org/ijicic-140507.pdf.
34. Genetic Algorithms: The Crossover-Mutation Debate [Електронний ресурс] – Режим доступу до ресурсу: https://www.semanticscholar.org/paper/Genetic-Algorithms%3A-The-Crossover-Mutation-Debate-Senaratna/73a50124700c7b2e44e3a72a298f6279a8b54ac3/figure/2.

35. Optimization of Train Trip Package Operation Scheme [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/figure/Mutation-operator-of-genetic-algorithm_fig4_282584731.

Додаток А
СТРУКТУРНА СХЕМА РОБОТИ
до дипломного проєкту
на тему: «Планування на основі штучного інтелекту»

Київ – 2020 року

СТРУКТУРНА СХЕМА РОБОТИ

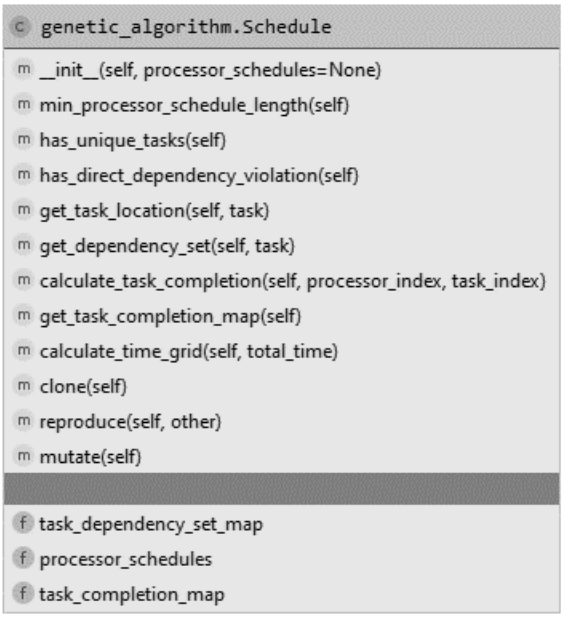


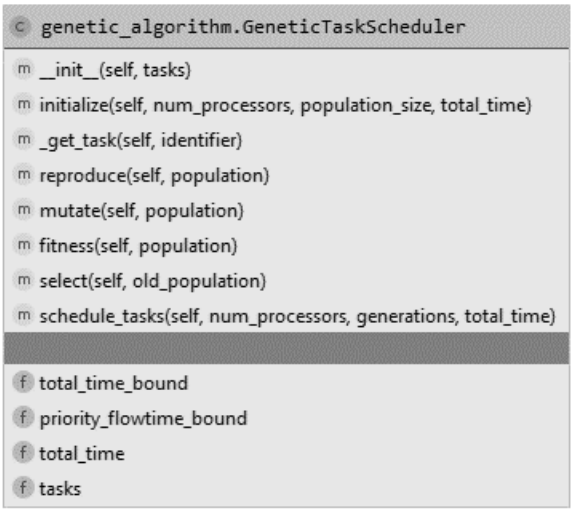
					ІАЛЦ.467200.004 Д1		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Кобилюк А.Г.			Планування на основі штучного інтелекту	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Волокита А.М.				Т	
Н.контр.		Сімоненко В.П.			Структурна схема роботи	НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІО-64	
Затв.		Волокита А.М.					

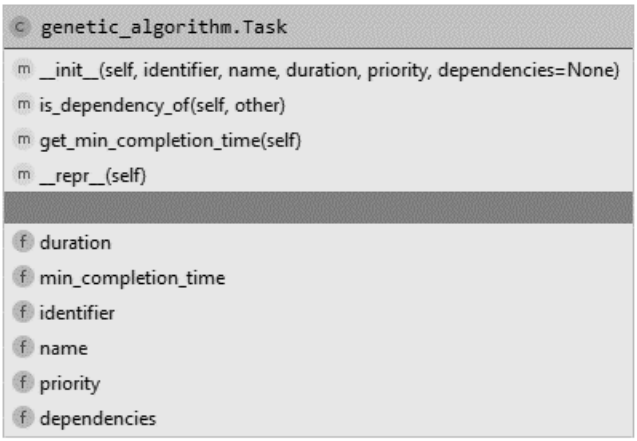
Додаток Б
ФУНКЦІОНАЛЬНА СХЕМА КЛАСІВ
до дипломного проєкту
на тему: «Планування на основі штучного інтелекту»

Київ – 2020 року

ФУНКЦІОНАЛЬНА СХЕМА КЛАСІВ



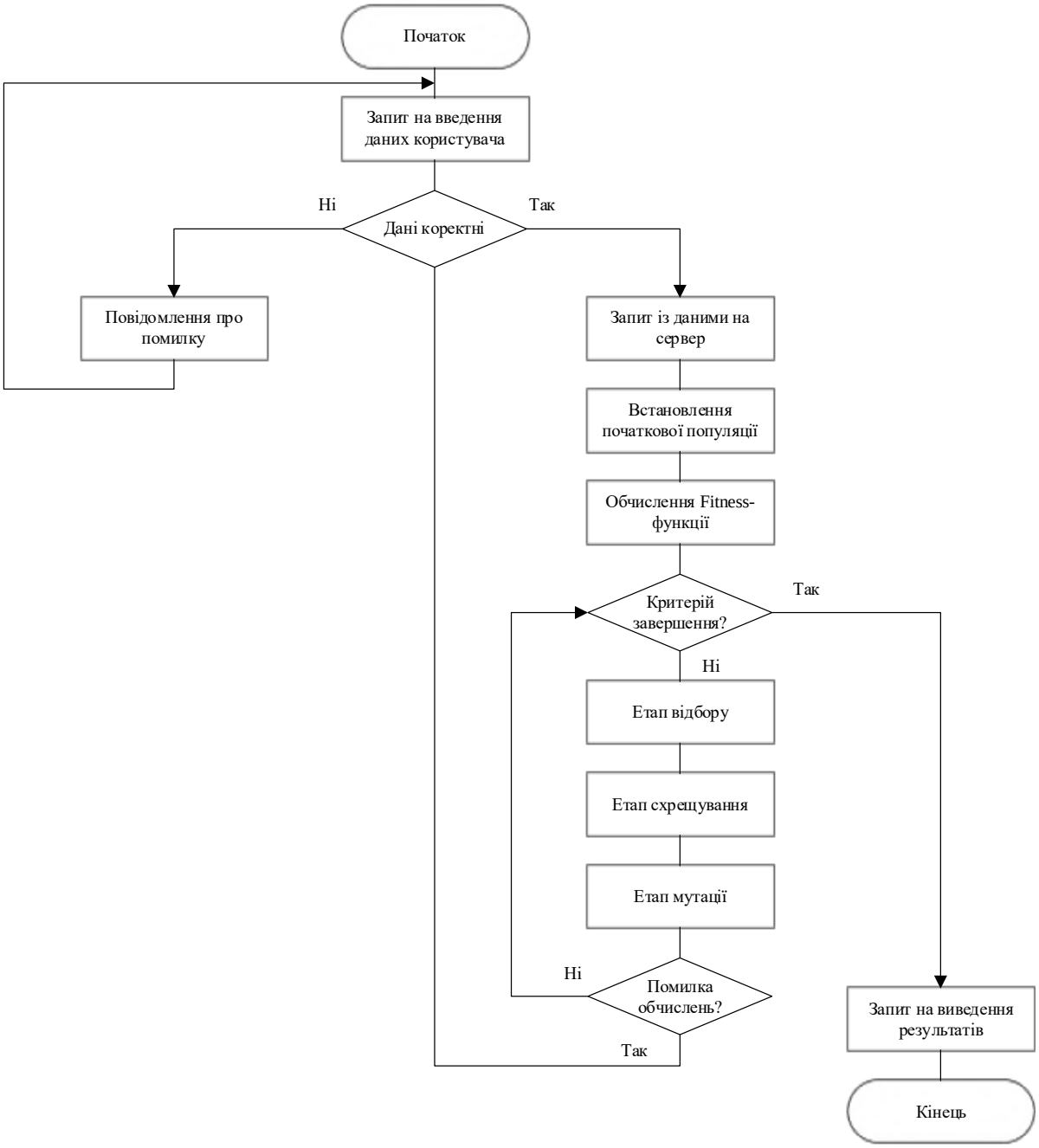




Додаток В
ПРИНЦИПОВА СХЕМА РОБОТИ
до дипломного проєкту
на тему: «Планування на основі штучного інтелекту»

Київ – 2020 року

ПРИНЦИПОВА СХЕМА РОБОТИ



					ІАЛЦ.467200.006 ДЗ	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		*Кобилюк А.Г.*			*Планування на основі штучного інтелекту*	*Літ.* / *Аркуш* / *Аркушів*
Перевірив		*Волокита А.М.*				Т / 1 / 1
Н.контр.		*Сімоненко В.П.*			*Принципова схема роботи*	*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІО-64*
Затв.		*Волокита А.М.*				

Додаток Г
ЛІСТИНГ ПРОГРАМИ
до дипломного проєкту
на тему: «Планування на основі штучного інтелекту»

Київ – 2020 року

ЛІСТИНГ ПРОГРАМИ

```
from flask import Flask, render_template, request
import json

import genetic_algorithm

app = Flask(__name__)

COLORS = ['#3498db', '#f1c40f', '#c0392b', '#a061ba', '#e67e22', '#1abc9c']

@app.route('/')
def main():
    return render_template('main.html')

@app.route('/schedule', methods=['POST'])
def schedule():
    data = request.json

    raw_tasks = data.get('tasks')
    tasks = []
    for task in raw_tasks:
        dependencies = task.get('depend')
        if dependencies:
            dependencies = [dependencies]

        t = genetic_algorithm.Task(task.get('id'), task.get('name'),
task.get('length'), task.get('priority'), dependencies)
        tasks.append(t)

    # Setup dependencies
    for task in tasks:
        dependencies = []
        for depend_id in task.dependencies:
            for t2 in tasks:
                if t2.identifier == depend_id:
                    dependencies.append(t2)
```

```
break
            else:
                raise Exception('Invalid dependency')
        task.dependencies = dependencies

    constraints = data.get('constraints')

    processors = constraints.get('processors')
    generations = constraints.get('generations')
    total_time = constraints.get('total_time')

    gen_alg = genetic_algorithm.GeneticTaskScheduler(tasks)
    schedule = gen_alg.schedule_tasks(processors, generations, total_time)

    # Reformat tasks in a way compatible with UI
    for i, processor in enumerate(schedule):
        print("\nP{} ".format(i), end="")
        for j, task in enumerate(processor):
            if task:
                schedule[i][j] = {'name': task.name, 'color':
COLORS[task.identifier % len(COLORS)]}
                print("T{}".format(j), end=" ")
            else:
                schedule[i][j] = None
                print("X".format(j), end=" ")
    print()
    return json.dumps(schedule)

if __name__ == "__main__":
    app.run(debug=True)

###############
#   imports
###############

import random

###############
#  Constants
###############

POPULATION_SIZE = 10
```

```
REPRODUCE_PROBABILITY = 0.5
MUTATION_PROBABILITY = 0.1

TOTAL_TIME_WEIGHT = 0.8
PRIORITY_FLOWTIME_WEIGHT = 0.2

###############
#   Classes
###############


class Task:
    def __init__(self, identifier, name, duration, priority,
dependencies=None):
        if not dependencies:
            dependencies = []

        self.identifier = identifier
        self.name = name
        self.duration = duration
        self.priority = priority
        self.dependencies = dependencies
        self.min_completion_time = -1

    def is_dependency_of(self, other):
        """
        Returns true if this task is a dependency (direct or indirect) of
other
        """
        return (self in other.dependencies) or
any(self.is_dependency_of(task) for task in other.dependencies)

    def get_min_completion_time(self):
        """
        Returns the minimum completion time of this task based on the minimum
completion times of its dependencies
        """
        if self.min_completion_time < 0:
            self.min_completion_time = self.duration
            if len(self.dependencies) > 0:
                self.min_completion_time += max(
                    [task.get_min_completion_time() for task in
```

```
self.dependencies])
        return self.min_completion_time

    def __repr__(self):
        return '<Task %s>' % self.identifier


class Schedule:
    def __init__(self, processor_schedules=None):
        if not processor_schedules:
            processor_schedules = []
        self.processor_schedules = processor_schedules
        self.task_completion_map = {}
        self.task_dependency_set_map = {}

    def min_processor_schedule_length(self):
        """
        Returns the number of tasks in the smallest processor
        """
        return min([len(processor) for processor in
self.processor_schedules])

    def has_unique_tasks(self):
        """
        Returns true if the schedule does not have more than one of the same
task
        """
        task_set = set()
        task_sum = 0
        for processor in self.processor_schedules:
            task_set.update(processor)
            task_sum += len(processor)
        return len(task_set) == task_sum

    def has_direct_dependency_violation(self):
        """
        Returns true if in any processor, a task is executed before one of
its dependencies
        """
        for processor in self.processor_schedules:
            for i in range(len(processor)):
                for dependency in processor[i].dependencies:
```

```
 if dependency in processor[i+1:]:
                        return True
        return False

    def get_task_location(self, task):
        """
        Returns the processor and task indices of any given task, None if the
task is not in this schedule
        """
        for i in range(len(self.processor_schedules)):
            for j in range(len(self.processor_schedules[i])):
                if self.processor_schedules[i][j].name == task.name:
                    return i, j
        return None

    def get_dependency_set(self, task):
        """
        Returns the set of all tasks the given task must be executed after in
this schedule
        """
        if task in self.task_dependency_set_map:
            return self.task_dependency_set_map[task]

        dependency_set = set()

        for dependency in task.dependencies:
            dependency_set.add(dependency)
            dependency_set.update(self.get_dependency_set(dependency))

        location = self.get_task_location(task)
        for previous_task in
self.processor_schedules[location[0]][0:location[1]]:
            dependency_set.add(previous_task)
            dependency_set.update(self.get_dependency_set(previous_task))

        self.task_dependency_set_map[task] = dependency_set

        return dependency_set

    def calculate_task_completion(self, processor_index, task_index):
        """
```

```
        Recursively determines when a task completes
        """
        task = self.processor_schedules[processor_index][task_index]

        if task in self.task_completion_map:
            return self.task_completion_map[task.identifier]

        previous_task_completion = 0 if task_index <= 0 else
self.calculate_task_completion(processor_index,

task_index - 1)
        dependency_completions = []
        for dependency in task.dependencies:
            if dependency in self.task_completion_map:

dependency_completions.append(self.task_completion_map[dependency])
            else:
                location = self.get_task_location(dependency)
                dependency_completions.append(
                    self.calculate_task_completion(location[0], location[1]))

        self.task_completion_map[task.identifier] =
max(previous_task_completion,

max(dependency_completions + [0])) + task.duration
        return self.task_completion_map[task.identifier]

    def get_task_completion_map(self):
        """
        For each task determine when it completes
        """
        for i in range(len(self.processor_schedules)):
            for j in range(len(self.processor_schedules[i])):
                self.calculate_task_completion(i, j)
        return self.task_completion_map

    def calculate_time_grid(self, total_time):
        """
        Calculates the grid of time slots and which task is executing during
each
        """
        time_grid = [[0 for x in range(total_time)] for y in
```

```
range(len(self.processor_schedules))]
        for i, processor in enumerate(self.processor_schedules):
            for j, task in enumerate(processor):
                end_time = self.calculate_task_completion(i, j)
                time_grid[i][(end_time - task.duration):end_time] = [task] *
task.duration

        return time_grid

    def clone(self):
        """
        Duplicates the schedule
        """
        return Schedule([list(processor) for processor in
self.processor_schedules])

    def reproduce(self, other):
        """
        Given another schedule, generates at most two offspring
        """
        max_crossover_index = min(self.min_processor_schedule_length(),
                                  other.min_processor_schedule_length())
        crossover_index = random.randrange(0, max_crossover_index + 1)

        child1 = self.clone()
        for i, processor in enumerate(child1.processor_schedules):
            processor[crossover_index:] =
other.processor_schedules[i][crossover_index:]

        child2 = other.clone()
        for i, processor in enumerate(child2.processor_schedules):
            processor[crossover_index:] =
self.processor_schedules[i][crossover_index:]

        return [child for child in [child1, child2] if not
child.has_direct_dependency_violation()]

    def mutate(self):
        """
        Chooses 1 task and moves it to a random valid index in the schedule
        """
        self.task_completion_map.clear()
```

```
self.task_dependency_set_map.clear()
        from_processor = random.choice([processor for processor in
self.processor_schedules if len(processor)])
        from_task = random.choice(from_processor)

        processor_range = list(range(len(self.processor_schedules)))
        random.shuffle(processor_range)
        for i in processor_range:
            to_processor = self.processor_schedules[i]

            min_to_index = 0
            for j in range(len(to_processor)):
                for dependency in from_task.dependencies:
                    if to_processor[j] == dependency or to_processor[j] in
self.get_dependency_set(dependency):
                        min_to_index = j + 1
                        break

            max_to_index = len(to_processor)
            for j in reversed(range(len(to_processor))):
                if from_task in self.get_dependency_set(to_processor[j]):
                    max_to_index = j

            if min_to_index <= max_to_index:
                insert_index = random.randrange(min_to_index, max_to_index +
1)
                if from_processor == to_processor and
from_processor.index(from_task) < insert_index:
                    to_processor.insert(insert_index, from_task)
                    from_processor.remove(from_task)
                else:
                    from_processor.remove(from_task)
                    to_processor.insert(insert_index, from_task)
                return

class GeneticTaskScheduler:
    def __init__(self, tasks):
        self.tasks = tasks
        self.total_time = 0
        self.total_time_bound = 0
        self.priority_flowtime_bound = 0
```

```
def initialize(self, num_processors, population_size, total_time):
        """
        This function, given the tasks, number of processors, and population
size, will
        produce an initial population.
        """
        self.total_time = total_time

        # Calculate upper bounds for fitness measures
        prioritized_tasks = list(self.tasks)
        prioritized_tasks.sort(key=lambda task: task.priority)
        for task in prioritized_tasks:
            self.total_time_bound += task.duration
            self.priority_flowtime_bound += self.total_time_bound *
task.priority

        self.total_time_bound += 1
        self.priority_flowtime_bound += 1

        # Generate first schedule based on min completion time
        self.tasks.sort(key=lambda task: task.get_min_completion_time())
        processor_schedules = [[] for i in range(num_processors)]

        for i in range(len(self.tasks)):
            processor_schedules[i % num_processors].append(self.tasks[i])

        population = [Schedule(processor_schedules)]

        # Generate the rest of the initial population from random mutations
of the first schedule
        for j in range(population_size - 1):
            new_schedule = population[0].clone()
            new_schedule.mutate()
            population.append(new_schedule)

        return population

    def _get_task(self, identifier):
        """
        Retrieves a task by its identifier
        """
        for task in self.tasks:
            if task.identifier == identifier:
```

```
                return task
        return None

    def reproduce(self, population):
        """
        This function, given the population, will randomly select individuals
for reproduction
        and add the children to the population
        """
        fertile_list = []
        for individual in population:
            if random.random() < REPRODUCE_PROBABILITY:
                fertile_list.append(individual)

        random.shuffle(fertile_list)
        for i in range(0, len(fertile_list) - 1, 2):
            population += fertile_list[i].reproduce(fertile_list[i + 1])

    def mutate(self, population):
        """
        This function, given the population, will randomly select individuals
for mutation
        """
        for individual in population:
            if individual.has_unique_tasks() and random.random() <
MUTATION_PROBABILITY:
                individual.mutate()

    def fitness(self, population):
        """
        This function calculates a list of fitness values for the population
        """
        fitness_list = []

        for schedule in population:
            if not schedule.has_unique_tasks():
                fitness_list.append(0)
                continue

            task_completions = schedule.get_task_completion_map()
            total_time = max(task_completions.values())
            if total_time > self.total_time:
                fitness_list.append(0)
```

```
                continue

            priority_flowtime = 0
            for key in task_completions:
                task = self._get_task(key)
                value = task_completions[key]
                priority_flowtime += task.priority * value

            fitness_value = round(TOTAL_TIME_WEIGHT * (self.total_time_bound
- total_time) +
                                  PRIORITY_FLOWTIME_WEIGHT *
(self.priority_flowtime_bound - priority_flowtime))
            fitness_list.append(fitness_value)

        return fitness_list

    def select(self, old_population):
        """
        This function randomly selects the individuals that survive to
reproduce based on their fitness
        """
        new_population = []
        fitness_list = self.fitness(old_population)
        fitness_sum = sum([val for val in fitness_list if val])

        for i in range(POPULATION_SIZE):
            survival_value = random.randrange(1, fitness_sum + 1)
            for j, fitness in enumerate(fitness_list):
                survival_value -= fitness
                if survival_value <= 0:
                    new_population.append(old_population[j])

        return new_population

    def schedule_tasks(self, num_processors, generations, total_time):
        """
        Given a list of constraints, this function will run the genetic
algorithm
        on the tasks to find a good schedule.
        """
        population = self.initialize(num_processors, POPULATION_SIZE,
total_time)
```

```
        for i in range(generations):
            self.select(population)
            self.reproduce(population)
            self.mutate(population)

        random.shuffle(population)
        fitness_list = self.fitness(population)
        best_schedule = population[fitness_list.index(max(fitness_list))]

        time_grid = []
        for processor in best_schedule.calculate_time_grid(total_time):
            time_grid.append(processor)

        return time_grid
```

```
<!DOCTYPE html>
<html ng-app="genAlg">{% raw %}
<head>
    <title>A Genetic Algorithm for Task Scheduling</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link href="../static/css/bootstrap.min.css" rel="stylesheet">
    <link href="../static/css/app.css" rel="stylesheet">
</head>
<body ng-controller="AppController">
<div class="row" id="header">
    <div class="col-lg-8">
        <h1>A Genetic Algorithm for Task Scheduling</h1>
    </div>
    <div class="col-lg-2 col-lg-offset-2">
        <h4><a href="https://github.com/Sanverik">Andrii Kobyliuk</a></h4>
        <a class="btn"
href="https://github.com/Sanverik/SchedulerAI">Repository</a>
    </div>
</div>

<div class="container">
    <div class="row">
        <div class="col-lg-5">

            <div class="row">
                <div class="col-lg-4">
                    <h5>Generations:</h5>
```

```
                    <input class="form-control" type="number" min="1"
max="20" ng-model="constraints.generations">
                </div>
                <div class="col-lg-4">
                    <h5>Processors:</h5>
                    <input class="form-control" type="number" min="1"
max="12" ng-model="constraints.processors" ng-change="updateSchedule()">
                </div>
                <div class="col-lg-4">
                    <h5>Total Time:</h5>
                    <input class="form-control" type="number" min="1"
max="24" ng-model="constraints.total_time" ng-change="updateSchedule()">
                </div>
            </div>

            <hr>

            <div id="task-input-container" class="row">
                <div class="row">
                    <div class="col-lg-3"><h5>Name</h5></div>
                    <div class="col-lg-3"><h5>Priority</h5></div>
                    <div class="col-lg-3"><h5>Length</h5></div>
                    <div class="col-lg-3"><h5>Dependency</h5></div>
                </div>
                <div ng-repeat="task in tasks" class="row task-input">
                    <div class="col-lg-3">
                        <input type="text" class="form-control" ng-
model="task.name">
                    </div>
                    <div class="col-lg-3">
                        <input type="number" class="form-control" ng-
model="task.priority">
                    </div>
                    <div class="col-lg-3">
                        <input type="number" class="form-control" ng-
model="task.length">
                    </div>
                    <div class="col-lg-3">
                        <div class="btn-group" style="width: 100%">
                            <button type="button" style="width: 100%"
class="btn btn-default dropdown-toggle" data-toggle="dropdown" ng-
disabled="tasks.length < 2">
```

```
                                {{ getDependency(task).name }} <span
class="caret"></span>
                            </button>
                            <ul class="dropdown-menu" role="menu">
                                <li ng-click="setDependency(task, null)" ng-
show="task.depend">
                                    <a href="#">None</a>
                                </li>
                                <li class="divider" ng-
show="task.depend"></li>
                                <li ng-repeat="depend in tasks" ng-
hide="task.depend == depend.id || task.id == depend.id || depend.depend ==
task.id" ng-click="setDependency(task, depend.id)">
                                    <a href="#" ng-
bind="depend.name"></a></li>
                            </ul>
                        </div>
                    </div>
                </div>
                <div class="row">
                    <div class="col-lg-6 col-lg-offset-3">
                        <br>
                        <button class="btn btn-block btn-default pull-right"
ng-click="addTask()">
                            Add Task
                        </button>
                    </div>
                </div>
            </div>
        </div>

        <div class="col-lg-6 col-lg-offset-1">
            <div class="row" id="schedule">
                <table class="table table-bordered table-responsive table-
hover">
                    <thead>
                    <tr>
                        <th class="processor-label">Processor</th>
                        <th class="time-label" ng-repeat="n in
range(schedule[0].length)">{{ $index
                            + 1}}
                        </th>
                    </tr>
```

```
                    </thead>
                    <tr ng-repeat="processor in schedule">
                        <td class="processor-label" ng-bind="$index +
1"></td>
                        <td ng-repeat="task in processor" ng-
style="getColor(task)" ng-bind="task.name"></td>
                    </tr>
                </table>
            </div>

            <div class="row">
                <button class="btn btn-primary btn-lg pull-right"
id="generate-schedule" ng-click="generateSchedule()">
                    Generate
                    Schedule
                </button>
            </div>
        </div>
    </div>
</div>

<script src="../static/js/lib/angular.min.js"></script>
<script src="../static/js/lib/jquery.min.js"></script>
<script src="../static/js/lib/bootstrap.min.js"></script>

<script src="../static/js/app.js"></script>

<script>


</script>
</body>
</html>
{% endraw %}
```